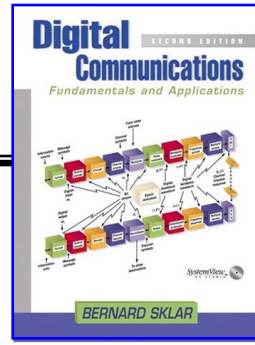


DIGITAL COMMUNICATIONS

Fundamentals and Applications

Second Edition

BERNARD SKLAR

Communications Engineering Services, Tarzana, California
and
University of California, Los Angeles

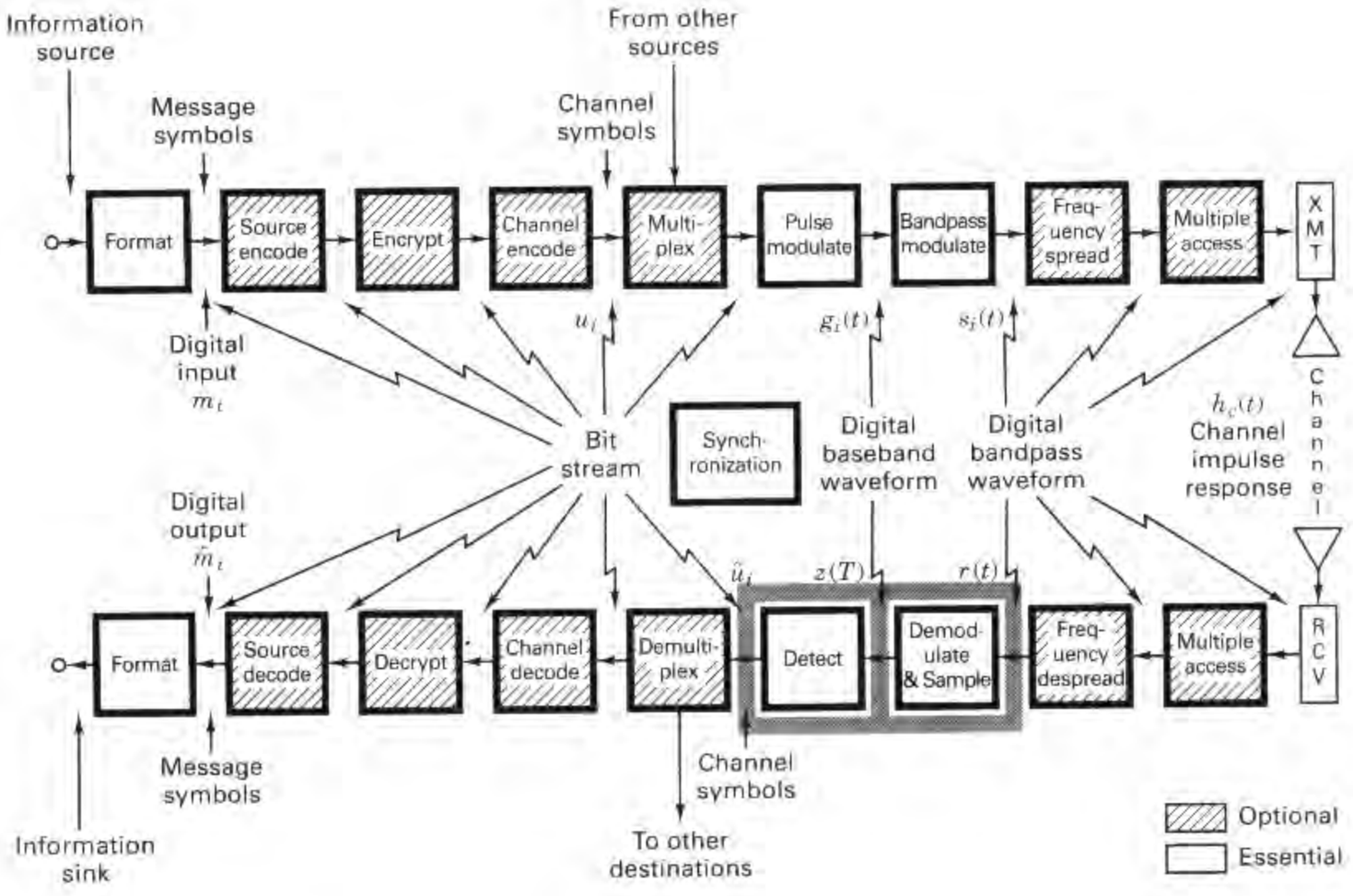
PH PTR

Prentice Hall P T R
Upper Saddle River, New Jersey 07458
www.phptr.com

ISBN 0-13-084788-7
90000
9 780130 847881

CHAPTER 6

Channel Coding: Part 1



Channel coding refers to the class of signal transformations designed to improve communications performance by enabling the transmitted signals to better withstand the effects of various channel impairments, such as noise, interference, and fading. These signal-processing techniques can be thought of as vehicles for accomplishing desirable system trade-offs (e.g., error-performance versus bandwidth, power versus bandwidth). Why do you suppose channel coding has become such a popular way to bring about these beneficial effects? The use of large-scale integrated circuits (LSI) and high-speed digital signal processing (DSP) techniques have made it possible to provide as much as 10 dB performance improvement through these methods, at much less cost than through the use of most other methods such as higher power transmitters or larger antennas.

6.1 WAVEFORM CODING

Channel coding can be partitioned into two study areas, waveform (or signal design) coding and structured sequences (or structured redundancy), as shown in Figure 6.1. *Waveform coding* deals with transforming waveforms into "better waveforms," to make the detection process less subject to errors. *Structured sequences* deals with transforming data sequences into "better sequences," having structured redundancy (redundant bits). The redundant bits can then be used for the detection and correction of errors. The encoding procedure provides the coded signal (whether waveforms or structured sequences) with better distance properties

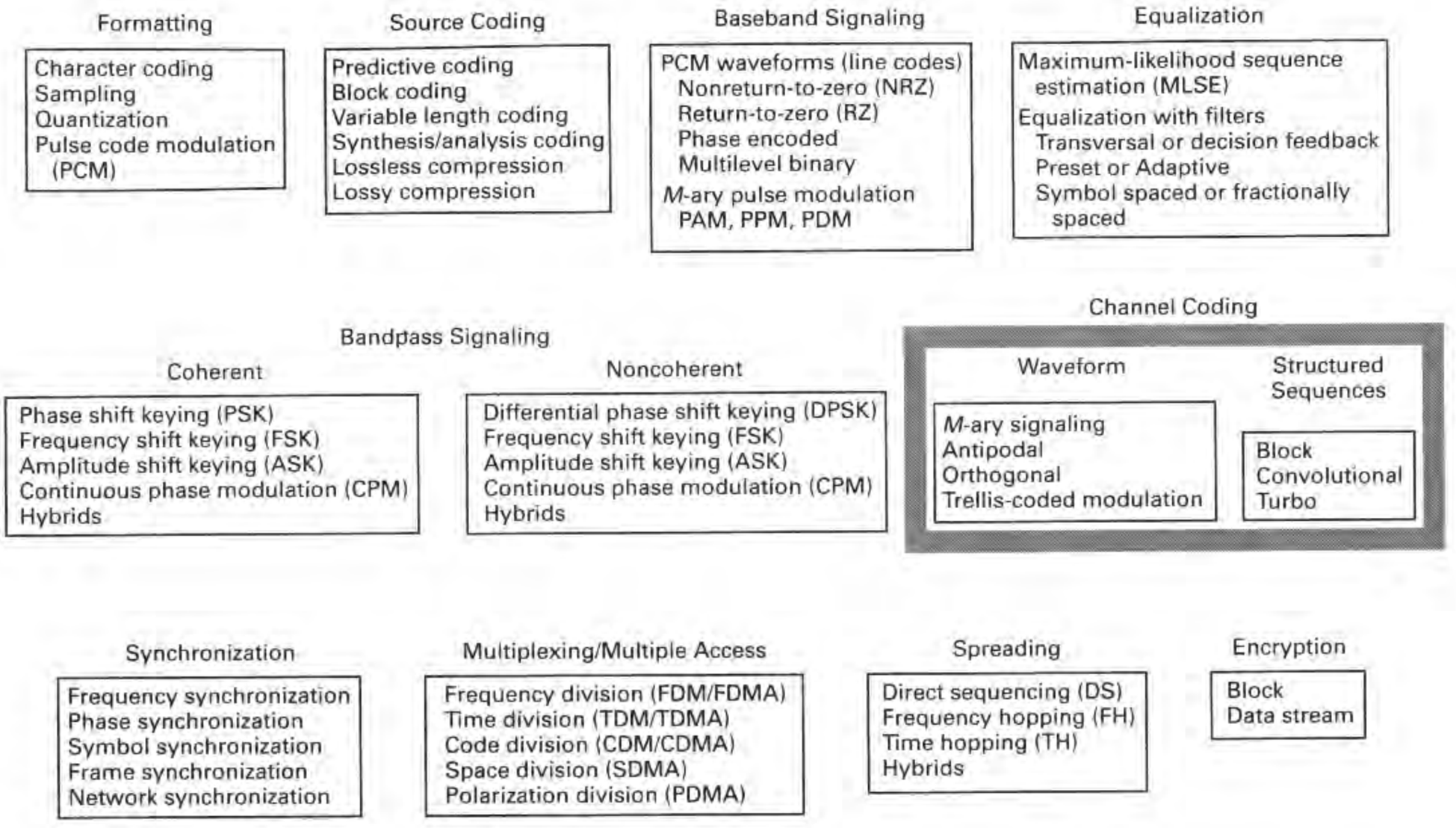


Figure 6.1 Basic digital communication transformations.

than those of their uncoded counterparts. First, we consider some waveform coding techniques. Then, starting with Section 6.3, we treat the subject of structured sequences.

6.1.1 Antipodal and Orthogonal Signals

Antipodal and orthogonal signals have been discussed earlier; we shall repeat the paramount features of these signal classes. The example shown in Figure 6.2 illustrates the analytical representation, $s_1(t) = -s_2(t) = \sin \omega_0 t$, $0 \le t \le T$, of a sinusoidal antipodal signal set, as well as its waveform representation and its vector representation. What are some synonyms or analogies that are used to describe *antipodal signals*? We can say that such signals are mirror images, or that one signal is the negative of the other, or that the signals are 180° apart.

The example shown in Figure 6.3 illustrates an orthogonal signal set made up of pulse waveforms, described by

$$s_1(t) = p(t) \qquad 0 \le t \le T$$

and

$$s_2(t) = p\left(t - \frac{T}{2}\right) \qquad 0 \le t \le T$$

where $p(t)$ is a pulse with duration $\tau = T/2$, and T is the symbol duration. Another orthogonal waveform set frequently used in communication systems is $\sin x$ and $\cos x$. In general, a set of equal-energy signals $s_i(t)$, where $i = 1, 2, \ldots, M$, constitutes an orthonormal (orthogonal, normalized to unity) set if and only if

$$z_{ij} = \frac{1}{E}\int_0^T s_i(t)s_j(t)\,dt = \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases} \tag{6.1}$$

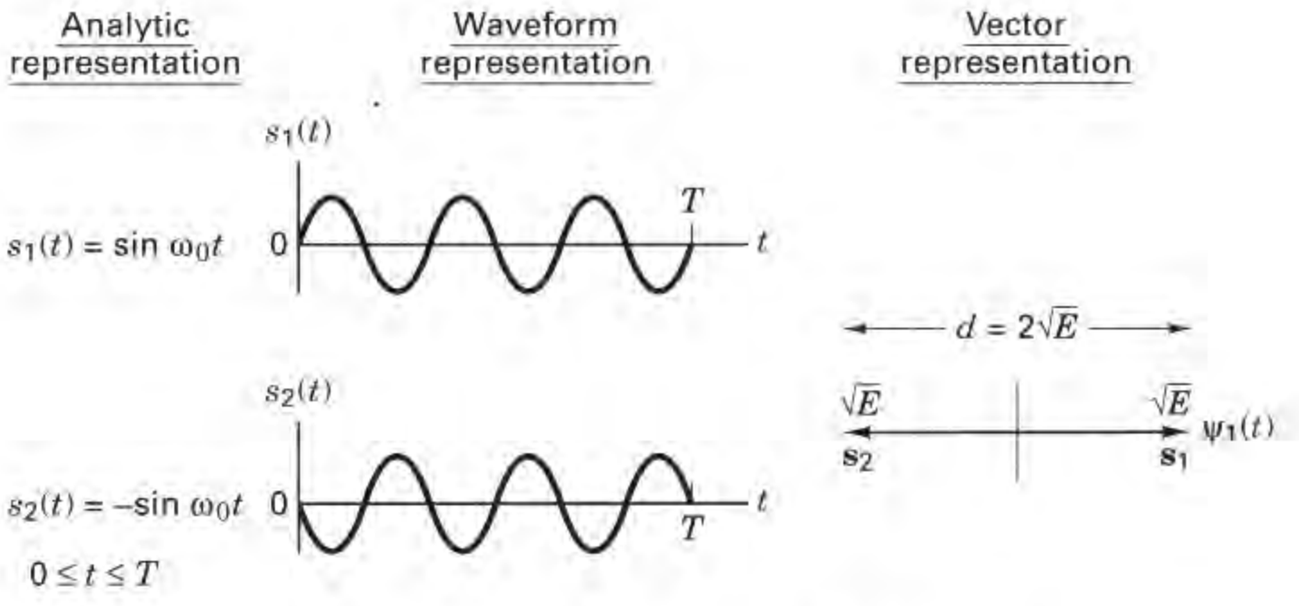


Figure 6.2 Example of an antipodal signal set.

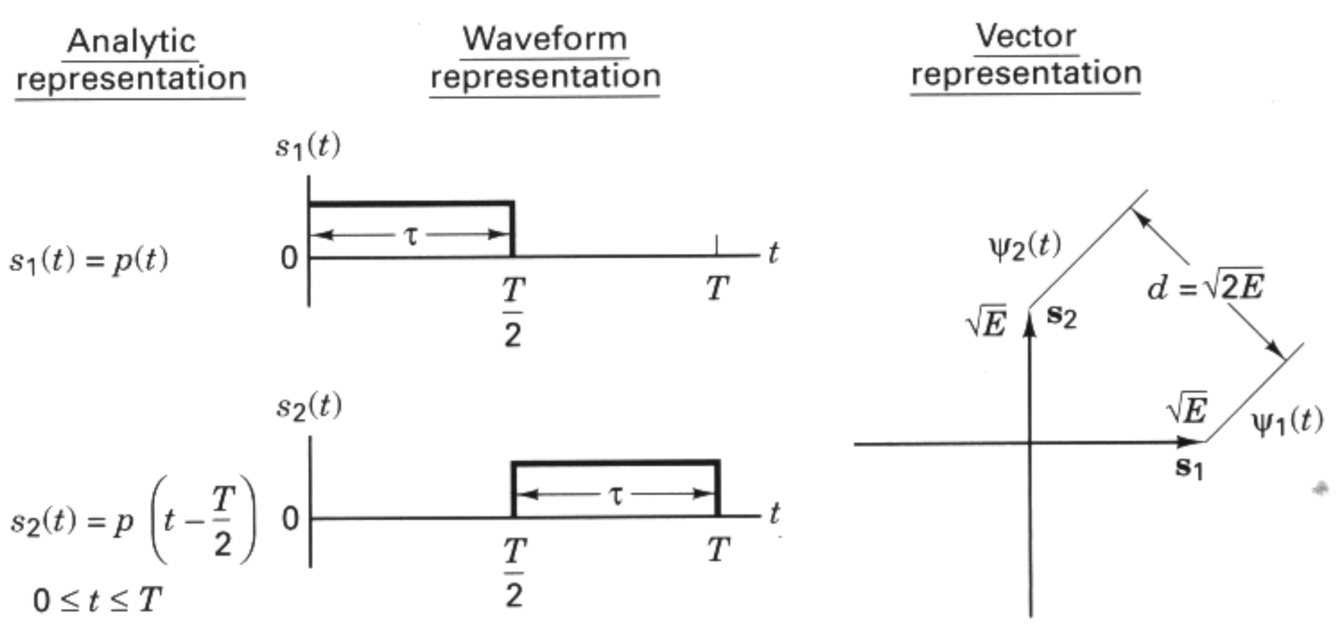


Figure 6.3 Example of a binary orthogonal signal set.

where z_{ij} is called the *cross-correlation coefficient*, and where E is the signal energy, expressed as

$$E = \int_0^T s_i^2(t)\, dt \tag{6.2}$$

The waveform representation in Figure 6.3 illustrates that $s_1(t)$ and $s_2(t)$ cannot interfere with one another because they are disjoint in time. The vector representation illustrates the perpendicular relationship between orthogonal signals. Consider some alternative descriptions of orthogonal signals or vectors. We can say that the inner or dot product of two different vectors in the orthogonal set must equal zero. In a two- or three-dimensional Cartesian coordinate space, we can describe the signal vectors, geometrically, as being mutually perpendicular to one another. We can say that one vector has zero projection on the other, or that one signal cannot interfere with the other, since they do not share the same *signal space*.

6.1.2 *M*-ary Signaling

With M-ary signaling, the processor accepts k data bits at a time. It then instructs the modulator to produce one of $M = 2^k$ waveforms; binary signaling is the special case where $k = 1$. For $k > 1$, M-ary signaling alone can be regarded as a *waveform coding* procedure. For orthogonal signaling (e.g., MFSK), as k increases there is improved error performance or a reduction in required E_b/N_0, at the expense of bandwidth; nonorthogonal signaling (e.g., MPSK) manifests improved bandwidth efficiency, at the expense of degraded error performance or an increase in required E_b/N_0. By the appropriate choice of signal waveforms, one can trade off error

probability, E_b/N_0, and bandwidth efficiency. Such trade-offs are treated in greater detail in Chapter 9.

6.1.3 Waveform Coding

Waveform coding procedures transform a waveform set (representing a message set) into an improved waveform set. The improved waveform set can then be used to provide improved P_B compared to the original set. The most popular of such *waveform codes* are referred to as *orthogonal* and *biorthogonal codes*. The encoding procedure endeavors to make each of the waveforms in the coded signal set as unalike as possible; the goal is to render the cross-correlation coefficient z_{ij}—among all pairs of signals, as described by the integral term in Equation 6.1—as small as possible. The smallest possible value of the cross-correlation coefficient occurs when the signals are anticorrelated ($z_{ij} = -1$); however, this can be achieved only when the number of symbols in the set is two ($M = 2$) and the symbols are *antipodal*. In general, it is possible to make all the cross-correlation coefficients equal to zero [1]. The set is then said to be *orthogonal*. Antipodal signal sets are optimum in the sense that each signal is most distant from the other signal in the set; this is seen in Figure 6.2 where the distance d between signal vectors is seen to be $d = 2\sqrt{E}$, where E represents the signal energy during a symbol duration T, as expressed in Equation (6.2). Compared with antipodal signals, the distance properties of orthogonal signal sets can be thought of as "pretty good" (for a given level of waveform energy). In Figure 6.3 the distance between the orthogonal signal vectors is seen to be $d = \sqrt{2E}$.

The *cross-correlation* between two signals is a measure of the *distance* between the signal vectors. The smaller the cross-correlation, the more distant are the vectors from each other. This can be verified in Figure 6.2, where the antipodal signals (whose $z_{ij} = -1$) are represented by vectors that are most distant from each other; and in Figure 6.3, where the orthogonal signals (whose $z_{ij} = 0$) are represented by vectors that are closer to one another than the antipodal vectors. It should be obvious that the distance between two identical waveforms (whose $z_{ij} = 1$) is zero.

The orthogonality condition in Equation (6.1) is presented in terms of waveforms $s_i(t)$ and $s_j(t)$, where $i, j = 1, \ldots, M$, and M is the size of the waveform set. Each waveform in the set $\{s_i(t)\}$ may consist of a sequence of pulses, where each pulse is designated with a level +1 or −1, which in turn represents the binary digit 1 or 0, respectively. When the set is expressed in this way, Equation (6.1) can be simplified by stating that $\{s_i(t)\}$ constitutes an orthogonal set if and only if

$$z_{ij} = \frac{\text{number of digit agreements} - \text{number of digit disagreements}}{\text{total number of digits in the sequence}} \tag{6.3}$$

$$= \begin{cases} 1 & \text{for } i = j \\ 0 & \text{otherwise} \end{cases}$$

6.1.3.1 Orthogonal Codes

A one-bit data set can be transformed, using *orthogonal codewords* of two digits each, described by the rows of matrix $\mathbf{H}_1$ as follows:

Data set	Orthogonal codeword set
0	$\mathbf{H}_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ (6.4a)
1	

For this, and the following examples, use Equation (6.3) to verify the orthogonality of the codeword set. To encode a 2-bit data set, we extend the foregoing set both horizontally and vertically, creating matrix $\mathbf{H}_2$.

Data set	Orthogonal codeword set
0 0	$\mathbf{H}_2 = \left[\begin{array}{cc:cc} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ \hdashline 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{array}\right] = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_1 \\ \mathbf{H}_1 & \overline{\mathbf{H}_1} \end{bmatrix}$ (6.4b)
0 1	
1 0	
1 1	

The lower right quadrant is the complement of the prior codeword set. We continue the same construction rule to obtain an orthogonal set $\mathbf{H}_3$ for a 3-bit data set.

Data Set	Orthogonal codeword set
0 0 0	$\mathbf{H}_3 = \left[\begin{array}{cccc:cccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ \hdashline 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = \begin{bmatrix} \mathbf{H}_2 & \mathbf{H}_2 \\ \mathbf{H}_2 & \overline{\mathbf{H}_2} \end{bmatrix}$ (6.4c)
0 0 1	
0 1 0	
0 1 1	
1 0 0	
1 0 1	
1 1 0	
1 1 1	

In general, we can construct a codeword set $\mathbf{H}_k$, of dimension $2^k \times 2^k$, called a *Hadamard matrix,* for a k-bit data set from the $\mathbf{H}_{k-1}$ matrix, as follows:

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{H}_{k-1} & \mathbf{H}_{k-1} \\ \mathbf{H}_{k-1} & \overline{\mathbf{H}_{k-1}} \end{bmatrix} \tag{6.4d}$$

Each pair of words in each codeword set $\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3, \ldots, \mathbf{H}_k, \ldots$ has as many digit agreements as disagreements [2]. Hence, in accordance with Equation (6.3), $z_{ij} = 0$ (for $i \neq j$), and each of the sets is orthogonal.

Just as M-ary signaling with an orthogonal modulation format (such as MFSK) improves the P_B performance, waveform coding with an orthogonally constructed signal set, in combination with coherent detection, produces *exactly the same* improvement. For equally likely, equal-energy orthogonal signals, the probability of codeword (symbol) error, P_E, can be upper bounded as [2]

$$P_E(M) \leq (M-1)\, Q\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{6.5}$$

where the codeword set M equals 2^k, and k is the number of data bits per codeword. The function $Q(x)$ is defined by Equation (3.43), and $E_s = kE_b$ is the energy per codeword. For a fixed M, as E_b/N_0 is increased, the bound becomes increasingly tight; for $P_E(M) \leq 10^{-3}$, Equation (6.5) is a good approximation. For expressing the bit-error probability, we next use the relationship between P_B and P_E, given in Equation (4.112) and repeated here:

$$\frac{P_B(k)}{P_E(k)} = \frac{2^{k-1}}{2^k - 1} \qquad \text{or} \qquad \frac{P_B(M)}{P_E(M)} = \frac{M/2}{(M-1)} \tag{6.6}$$

Combining Equations (6.5) and (6.6), the probability of bit error can be bounded as follows:

$$P_B(k) \leq (2^{k-1})\, Q\left(\sqrt{\frac{kE_b}{N_0}}\right) \qquad \text{or} \qquad P_B(M) \leq \frac{M}{2} Q\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{6.7}$$

6.1.3.2 Biorthogonal Codes

A *biorthogonal* signal set of M total signals or codewords can be obtained from an orthogonal set of $M/2$ signals by augmenting it with the negative of each signal as follows:

$$\mathbf{B}_k = \begin{bmatrix} \mathbf{H}_{k-1} \\ \hline -\mathbf{H}_{k-1} \end{bmatrix}$$

For example, a 3-bit data set can be transformed into a biothogonal codeword set as follows:

Data set	Biorthogonal codeword set
0 0 0	$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ \hline 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix}$
0 0 1	
0 1 0	
0 1 1	
1 0 0	
1 0 1	
1 1 0	
1 1 1	

$\mathbf{B}_3 =$

The biorthogonal set is really two sets of orthogonal codes such that each codeword in one set has its antipodal codeword in the other set. The biorthogonal set consists of a *combination of orthogonal and antipodal* signals. With respect to z_{ij} of Equation (6.1), biorthogonal codes can be characterized as

$$z_{ij} = \begin{cases} 1 & \text{for } i = j \\ -1 & \text{for } i \neq j,\ |i - j| = \dfrac{M}{2} \\ 0 & \text{for } i \neq j,\ |i - j| \neq \dfrac{M}{2} \end{cases} \tag{6.8}$$

One advantage of a biorthogonal code over an orthogonal one for the same data set is that the biorthogonal code requires *one-half* as many code bits per codeword (compare the rows of the $\mathbf{B}_3$ matrix with those of the $\mathbf{H}_3$ matrix presented earlier). Thus the bandwidth requirements for biorthogonal codes are one-half the requirements for comparable orthogonal ones. Since antipodal signal vectors have better distance properties than orthogonal ones, it should come as no surprise that biorthogonal codes perform slightly better than orthogonal ones. For equally likely, equal-energy biorthogonal signals, the probability of codeword (symbol) error can be upper bounded, as follows [2]:

$$P_E(M) \leq (M - 2)Q\left(\sqrt{\frac{E_s}{N_0}}\right) + Q\left(\sqrt{\frac{2E_s}{N_0}}\right) \tag{6.9}$$

which becomes increasingly tight for fixed M as E_b/N_0 is increased. $P_B(M)$ is a complicated function of $P_E(M)$; we can approximate it with the relationship [2]

$$P_B(M) \approx \frac{P_E(M)}{2}$$

The approximation is quite good for $M > 8$. Therefore, we can write

$$P_B(M) \lesssim \tfrac{1}{2}\left[(M - 2)Q\left(\sqrt{\frac{E_s}{N_0}}\right) + Q\left(\sqrt{\frac{2E_s}{N_0}}\right)\right] \tag{6.10}$$

These biorthogonal codes offer improved P_B performance, compared with the performance of the orthogonal codes, and require only *half the bandwidth* of orthogonal codes.

6.1.3.3 Transorthogonal (Simplex) Codes

A code generated from an orthogonal set by deleting the first digit of each codeword is called a *transorthogonal* or *simplex code.* Such a code is characterized by

$$z_{ij} = \begin{cases} 1 & \text{for } i = j \\ \dfrac{-1}{M - 1} & \text{for } i \neq j \end{cases} \tag{6.11}$$

A simplex code represents the *minimum energy* equivalent (in the error-probability sense) of the equally likely orthogonal set. In comparing the error performance of orthogonal, biorthogonal, and simplex codes, we can state that simplex coding requires the minimum E_b/N_0 for a specified symbol error rate. However, for a *large value of M*, all three schemes are *essentially identical* in error performance. Biorthogonal coding requires half the bandwidth of the others. But for each of these codes, bandwidth requirements (and system complexity) grow exponentially with the value of M; therefore, such coding schemes are attractive only when large bandwidths are available.

6.1.4 Waveform-Coding System Example

Figure 6.4 illustrates an example of assigning a k-bit message from a message set of size $M = 2^k$, with a coded-pulse sequence from a code set of the same size. Each k-bit message chooses one of the generators yielding a coded-pulse sequence or codeword. The sequences in the coded set that replace the messages form a waveform set with good distance properties (e.g., orthogonal, biorthogonal). For the orthogonal code described in Section 6.1.3.1, each codeword consists of $M = 2^k$ pulses (representing code bits). Hence 2^k code bits replace k message bits. The chosen sequence then modulates a carrier wave using binary PSK, such that the phase ($\phi_j = 0$ or π) of the carrier during each code-bit duration, $0 \le t \le T_c$, corresponds to the amplitude ($j = -1$ or 1) of the jth bipolar pulse in the codeword. At the receiver in Figure 6.5, the signal is demodulated to baseband and fed to M correlators (or matched filters). For orthogonal codes, such as those characterized by the Hadamard matrix in Section 6.1.3.1, correlation is performed over a codeword duration that can be expressed as $T = 2^k T_c$. For a real-time communication system, messages may not be delayed; hence, the codeword duration must be the same as the message duration, and thus, T can also be expressed as $T = (\log_2 M)\, T_b = kT_b$.

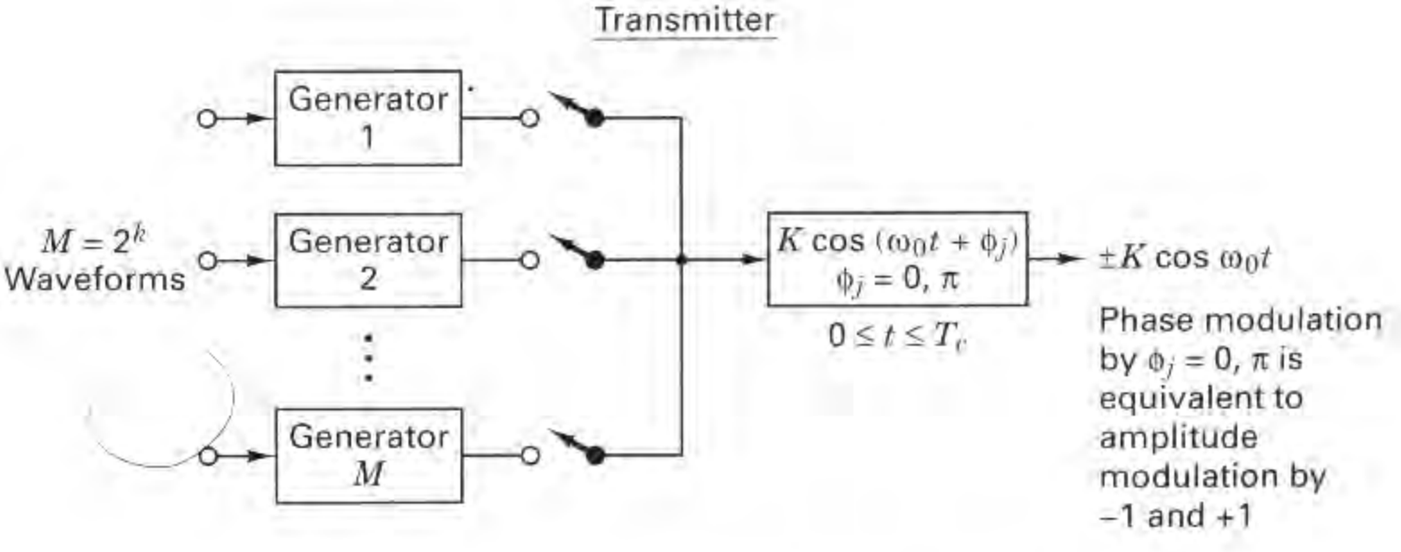


Figure 6.4 Waveform-encoded system (transmitter).

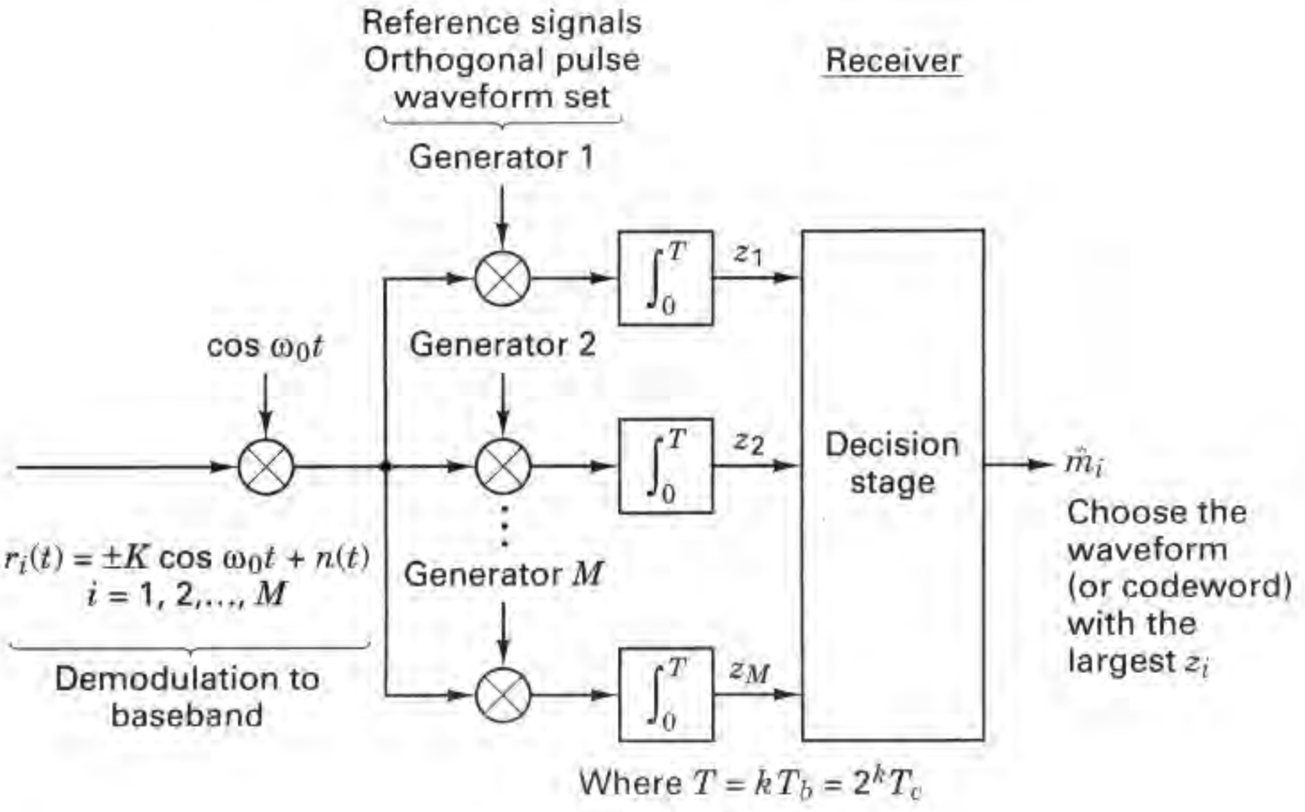


Figure 6.5 Waveform-encoded system with coherent detection (receiver).

where T_b is the message-bit duration. Note that the time duration of a message bit is M/k times longer than that of a code bit. In other words, the code bits or coded pulses (which are PSK modulated) must move at a rate M/k faster than the message bits. For such orthogonally coded waveforms and an AWGN channel, the expected value at the output of each correlator, at time T, is zero, except for the correlator corresponding to the transmitted codeword.

What is the advantage of such orthogonal waveform coding compared with simply sending one bit or one pulse at a time? One can compare the bit-error performance with and without such coding by comparing Equation (4.79) for coherent detection of antipodal signals with Equation (6.7) for the coherent detection of orthogonal codewords. For a given size k-bit message (say, $k = 5$) and a desired bit-error probability (say, 10^{-5}), the detection of orthogonal codewords (each having a 5-bit meaning) can be accomplished with about 2.9 dB less E_b/N_0 than the bit-by-bit detection of antipodal signals. (The demonstration is left as an exercise for the reader in Problem 6.28.) One might have guessed this result by comparing the performance curves for orthogonal signaling in Figure 4.28 with the binary (antipodal) curve in Figure 4.29. What price do we pay for this error-performance improvement? The cost is more transmission bandwidth. In this example, transmission of an uncoded message consists of sending 5 bits. With coding, how many coded pulses must be transmitted for each message sequence? With the waveform coding of this example, each 5-bit message sequence is represented by $M = 2^k = 2^5 = 32$ code bits or coded pulses. The 32 coded pulses in a codeword must be sent in the same time duration as the corresponding 5 bits from which they stem. Thus, the

required transmission bandwidth is 32/5 times that of the uncoded case. In general, the bandwidth needed for such orthogonally coded signals is M/k times greater than that needed for the uncoded case. Later, more efficient ways to trade off the benefits of coding versus bandwidth [3, 4] will be examined.

6.2 TYPES OF ERROR CONTROL

Before we discuss the details of structured redundancy, let us describe the two basic ways such redundancy is used for controlling errors. The first, *error detection and retransmission,* utilizes *parity bits* (redundant bits added to the data) to detect that an error has been made. The receiving terminal does not attempt to correct the error; it simply requests that the transmitter retransmit the data. Notice that a two-way link is required for such dialogue between the transmitter and receiver. The second type of error control, *forward error correction* (FEC), requires a one-way link only, since in this case the parity bits are designed for both the detection and correction of errors. We shall see that not all error patterns can be corrected; error-correcting codes are classified according to their error-correcting capabilities.

6.2.1 Terminal Connectivity

Communication terminals are often classified according to their connectivity with other terminals. The possible connections, shown in Figure 6.6, are termed *simplex* (not to be confused with the simplex or transorthogonal codes), *half-duplex*, and *full-duplex*. The simplex connection in Figure 6.6a is a one-way link. Transmissions

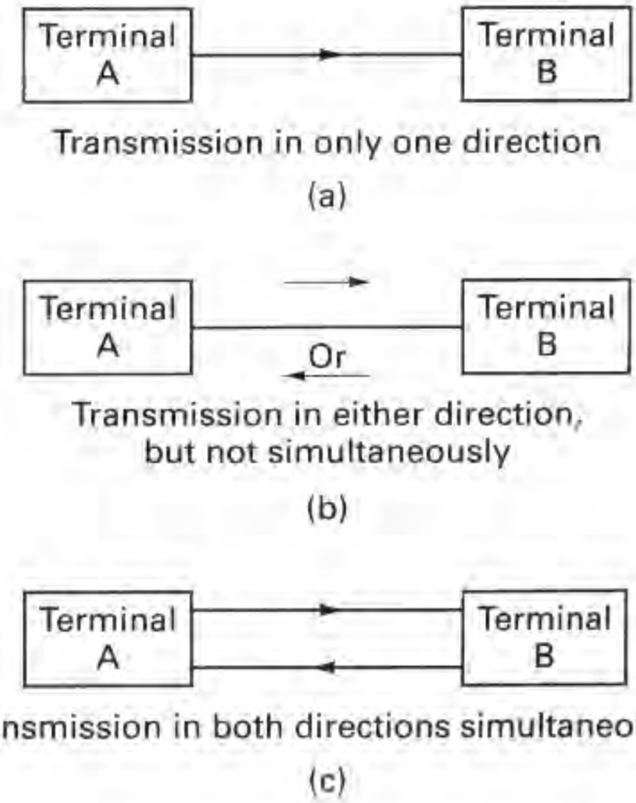


Figure 6.6 Terminal connectivity classifications. (a) Simplex. (b) Half-duplex. (c) Full-duplex.

are made from terminal A to terminal B only, never in the reverse direction. The half-duplex connection in Figure 6.6b is a link whereby transmissions may be made in either direction but not simultaneously. Finally, the full-duplex connection in Figure 6.6c is a two-way link, where transmissions may proceed in both directions simultaneously.

6.2.2 Automatic Repeat Request

When the error control consists of error detection only, the communication system generally needs to provide a means of alerting the transmitter that an error has been detected and that a retransmission is necessary. Such error control procedures are known as *automatic repeat request* or automatic retransmission query (ARQ) methods. Figure 6.7 illustrates three of the most popular ARQ procedures. In each of the diagrams, time is advancing from left to right. The first procedure, called *stop-and-wait ARQ*, is shown in Figure 6.7a. It requires a half-duplex connection only, since the transmitter waits for an acknowledgment (ACK) of each transmis-

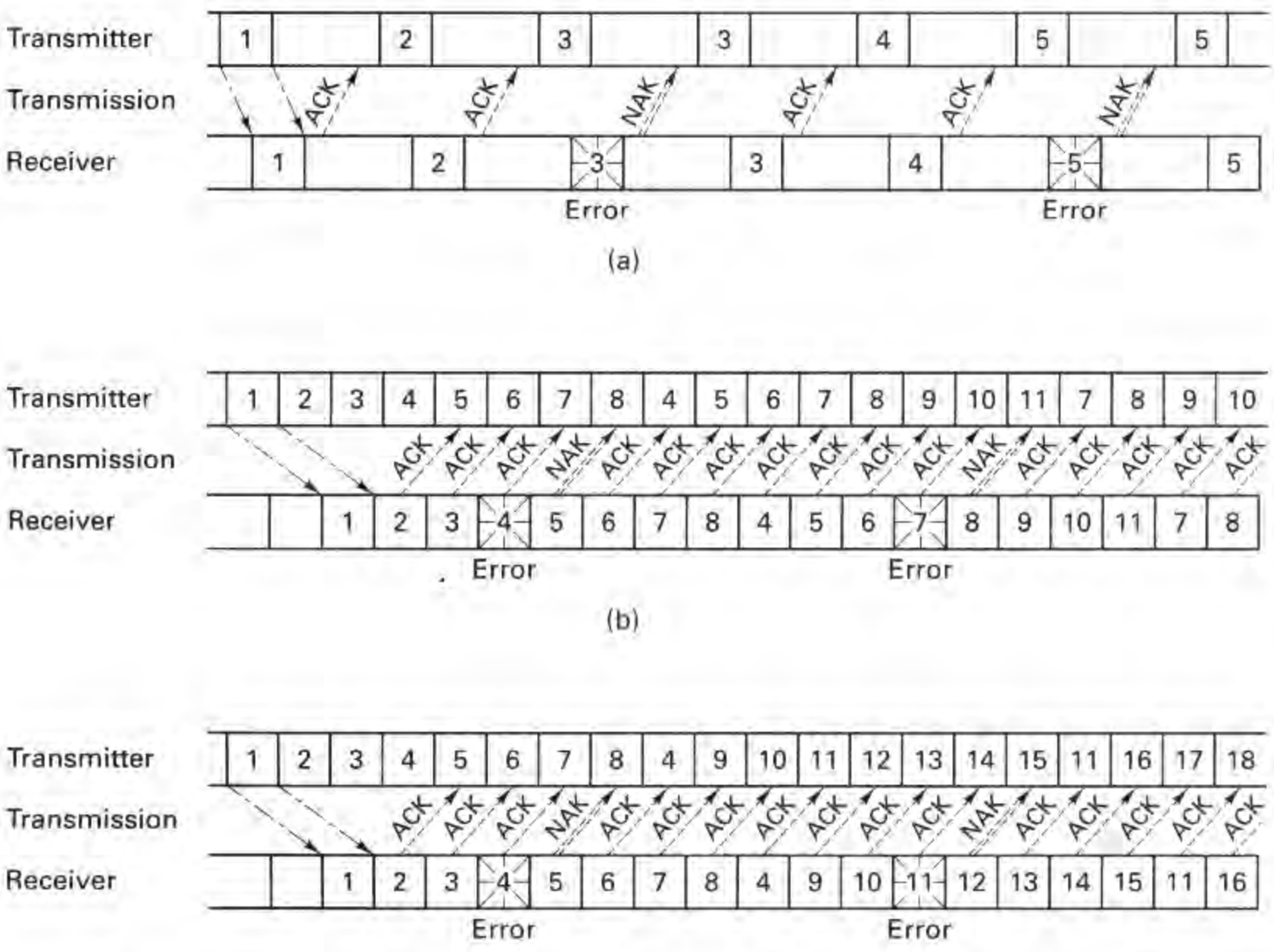


Figure 6.7 Automatic repeat request (ARQ). (a) Stop-and-wait ARQ (half-duplex). (b) Continuous ARQ with pullback (full-duplex). (c) Continuous ARQ with selective repeat (full-duplex).

sion before it proceeds with the next transmission. In the figure, the third transmission block is received in error; therefore, the receiver responds with a negative acknowledgment (NAK), and the transmitter retransmits this third message block before transmitting the next in the sequence. The second ARQ procedure, called *continuous ARQ with pullback*, is shown in Figure 6.7b. Here a full-duplex connection is necessary. Both terminals are transmitting simultaneously; the transmitter is sending message data and the receiver is sending acknowledgment data. Notice that a sequence number has to be assigned to each block of data. Also, the ACKs and NAKs need to reference such numbers, or else there needs to be *a priori* knowledge of the propagation delays, so that the transmitter knows which messages are associated with which acknowledgments. In the example of Figure 6.7b, there is a fixed separation of four blocks between the message being transmitted and the acknowledgment being simultaneously received. For example, when message 8 is being sent, a NAK corresponding to the corrupted message 4 is being received. In the ARQ procedure, the transmitter "pulls back" to the message in error and retransmits all message data, starting with the corrupted message. The final method, called *continuous ARQ with selective repeat*, is shown in Figure 6.7c. Here, as with the second ARQ procedure, a full-duplex connection is needed. In this procedure, however, only the corrupted message is repeated; then, the transmitter continues the transmission sequence where it had left off instead of repeating any subsequent correctly received messages.

The choice of which ARQ procedure to choose is a trade-off between the requirements for efficient utilization of the communications resource and the need to provide full-duplex connectivity. The half-duplex connectivity required in Figure 6.7a is less costly than ful-duplex; the associated inefficiency can be measured by the blank time slots. The more efficient utilization illustrated in Figures 6.7b and c requires the more costly full-duplex connectivity.

The major advantage of ARQ over forward error correction (FEC) is that error detection requires much simpler decoding equipment and much less redundancy than does error correction. Also, ARQ is adaptive in the sense that information is retransmitted only when errors occur. On the other hand, FEC may be desirable in place of, or in addition to, error detection, for any of the following reasons:

1. A reverse channel is not available or the delay with ARQ would be excessive.
2. The retransmission strategy is not conveniently implemented.
3. The expected number of errors, without corrections, would require excessive retransmissions.

6.3 STRUCTURED SEQUENCES

In Section 4.8 we considered digital signaling by means of $M = 2^k$ signal waveforms (M-ary signaling), where each waveform contains k bits of information. We saw that in the case of orthogonal M-ary signaling, we can decrease P_B by increasing M

(expanding the bandwidth). Similarly, in Section 6.1 we showed that it is possible to decrease P_B by encoding k binary digits into one of M orthogonal codewords. The major disadvantage with such orthogonal coding techniques is the associated inefficient use of bandwidth. For an orthogonally coded set of $M = 2^k$ waveforms, the required transmission bandwidth is M/k times that needed for the uncoded case. In this and subsequent sections we abandon the need for antipodal or orthogonal properties and focus on a class of encoding procedures known as *parity-check codes.* Such channel coding procedures are classified as *structured sequences* because they represent methods of inserting structured redundancy into the source data so that the presence of errors can be detected or the errors corrected. Structured sequences are partitioned into three subcategories, as shown in Figure 6.1: *block, convolutional,* and *turbo.* Block coding (primarily) is treated in this chapter, and the others are treated in Chapters 7 and 8 respectively.

6.3.1 Channel Models

6.3.1.1 Discrete Memoryless Channel

A *discrete memoryless channel* (DMC) is characterized by a discrete input alphabet, a discrete output alphabet, and a set of conditional probabilities $P(j|i)$ $(1 \le i \le M,\ 1 \le j \le Q)$, where i represents a modulator M-ary input symbol, j represents a demodulator Q-ary output symbol, and $P(j|i)$ is the probability of receiving j given that i was transmitted. Each output symbol of the channel depends only on the corresponding input, so that for a given input sequence $\mathbf{U} = u_1, u_2, \ldots, u_m, \ldots, u_N$, the conditional probability of a corresponding output sequence $\mathbf{Z} = z_1, z_2, \ldots, z_m, \ldots, z_N$ may be expressed as

$$P(\mathbf{Z}|\mathbf{U}) = \prod_{m=1}^{N} P(z_m|u_m) \tag{6.12}$$

In the event that the channel *has memory* (i.e., noise or fading that occurs in bursts), the conditional probability of the sequence $\mathbf{Z}$ would need to be expressed as the *joint* probability of all the elements of the sequence. Equation (6.12) expresses the *memoryless* condition of the channel. Since the channel noise in a memoryless channel is defined to affect each symbol independently of all the other symbols, the conditional probability of $\mathbf{Z}$ is seen as the product of the independent element probabilities.

6.3.1.2 Binary Symmetric Channel

A *binary symmetric channel* (BSC) is a special case of a DMC; the input and output alphabet sets consist of the binary elements (0 and 1). The conditional probabilities are symmetric:

$$P(0|1) = P(1|0) = p \tag{6.13}$$

and

$$P(1|1) = P(0|0) = 1 - p$$

Equation (6.13) expresses the channel *transition probabilities*. That is, given that a channel symbol was transmitted, the probability that it is received in error is p (related to the symbol energy), and the probability that it is received correctly is $(1-p)$. Since the demodulator output consists of the discrete elements 0 and 1, the demodulator is said to make a firm or *hard decision* on each symbol. A commonly used code system consists of BPSK modulated coded data and hard decision demodulation. Then the channel symbol error probability is found using the methods discussed in Section 4.7.1 and Equation (4.79) to be

$$p = Q\left(\sqrt{\frac{2E_c}{N_0}}\right)$$

where E_c/N_0 is the channel symbol energy per noise density, and $Q(x)$ is defined in Equation (3.43).

When such hard decisions are used in a binary coded system, the demodulator feeds the two-valued *code symbols* or *channel bits* to the decoder. Since the decoder then operates on the hard decisions made by the demodulator, decoding with a BSC channel is called *hard-decision decoding*.

6.3.1.3 Gaussian Channel

We can generalize our definition of the DMC to channels with alphabets that are not discrete. An example is the *Gaussian channel* with a discrete input alphabet and a continuous output alphabet over the range $(-\infty, \infty)$. The channel adds noise to the symbols. Since the noise is a Gaussian random variable with zero mean and variance σ^2, the resulting probability density function (pdf) of the received random variable z, conditioned on the symbol u_k (the likelihood of u_k), can be written as

$$p(z \mid u_k) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[\frac{-(z-u_k)^2}{2\sigma^2}\right] \tag{6.14}$$

for all z, where $k = 1, 2, \ldots, M$. For this case, *memoryless* has the same meaning as it does in Section 6.3.1.1, and Equation (6.12) can be used to obtain the conditional probability for the sequence $\mathbf{Z}$.

When the demodulator output consists of a continuous alphabet or its quantized approximation (with greater than two quantization levels), the demodulator is said to make *soft decisions*. In the case of a coded system, the demodulator feeds such quantized code symbols to the decoder. Since the decoder then operates on the soft decisions made by the demodulator, decoding with a Gaussian channel is called *soft-decision decoding*.

In the case of a hard-decision channel, we are able to characterize the detection process with a channel symbol error probability. However, in the case of a soft-decision channel, the detector makes the kind of decisions (soft decisions) that cannot be labeled as correct or incorrect. Thus, since there are no firm decisions, there cannot be a probability of making an error; the detector can only formulate a family of conditional probabilities or likelihoods of the different symbol types.

It is possible to design decoders using soft decisions, but block code soft-decision decoders are substantially more complex than hard-decision decoders; therefore, block codes are usually implemented with hard-decision decoders. For convolutional codes, both hard- and soft-decision implementations are equally popular. In this chapter we consider that the channel is a binary symmetric channel (BSC), and hence the decoder employs hard decisions. In Chapter 7 we further discuss channel models, as well as hard- versus soft-decision decoding for convolutional codes.

6.3.2 Code Rate and Redundancy

In the case of block codes, the source data are segmented into blocks of k data bits, also called information bits or message bits; each block can represent any one of 2^k distinct messages. The encoder transforms each k-bit data block into a larger block of n bits, called code bits or channel symbols. The $(n - k)$ bits, which the encoder adds to each data block, are called *redundant bits, parity bits,* or *check bits;* they carry no new information. The code is referred to as an (n, k) code. The ratio of redundant bits to data bits, denoted $(n - k)/k$, within a block is called the *redundancy* of the code; the ratio of data bits to total bits, k/n, is called the *code rate.* The code rate can be thought of as the portion of a code bit that constitutes information. For example, in a rate $\frac{1}{2}$ code, each code bit carries $\frac{1}{2}$ bit of information.

In this chapter and in Chapters 7 and 8 we consider those coding techniques that provide redundancy by increasing the required transmission bandwidth. For example, an error control technique that employs a rate 1/2 code (100% redundancy) will require double the bandwidth of an uncoded system. However, if a rate 3/4 code is used, the redundancy is 33% and the bandwidth expansion is only 4/3. In Chapter 9 we consider modulation/coding techniques for bandlimited channels where complexity instead of bandwidth is traded for error performance improvement.

6.3.2.1 Code-Element Nomenclature

Different authors describe an encoder's output elements in a variety of ways: code bits, channel bits, code symbols, channel symbols, parity bits, parity symbols. The terms are all very similar. In this text, for a binary code, the terms "code bit," "channel bit," "code symbol," and "channel symbol" have exactly the same meaning. The terms "code bit" and "channel bit" are most descriptive for binary codes only. The more generic names "code symbol" and "channel symbol" are often preferred because they can be used to describe binary or nonbinary codes equally well. Note that such code symbols or channel symbols are not to be confused with the grouping of bits to form transmission symbols that was done in previous chapters. The terms "parity bit" and "parity symbol" are used to identify only those code elements that represent the redundancy components added to the original data.

6.3.3 Parity-Check Codes

6.3.3.1 Single-Parity-Check Code

Parity-check codes use linear sums of the information bits, called *parity symbols* or *parity bits,* for error detection or correction. A single-parity-check code is constructed by adding a single-parity bit to a block of data bits. The parity bit takes on the value of one or zero as needed to ensure that the summation of all the bits in the codeword yields an even (or odd) result. The summation operation is performed using modulo-2 arithmetic (exclusive-or logic), as described in Section 2.9.3. If the added parity is designed to yield an even result, the method is termed *even parity;* if it is designed to yield an odd result, the method is termed *odd parity.* Figure 6.8a illustrates a serial data transmission (the rightmost bit is the earliest bit). A single-parity bit is added (the leftmost bit in each block) to yield even parity.

At the receiving terminal, the decoding procedure consists of testing that the modulo-2 sum of the codeword bits yields a zero result (even parity). If the result is found to be one instead of zero, the codeword is known to contain errors. The rate of the code can be expressed as $k/(k+1)$. Do you suppose the decoder can automatically *correct* a digit that is received in error? No, it cannot. It can only *detect* the presence of an odd number of bit errors. (If an even number of bits are inverted, the parity test will appear correct, which represents the case of an *undetected error.*) Assuming that

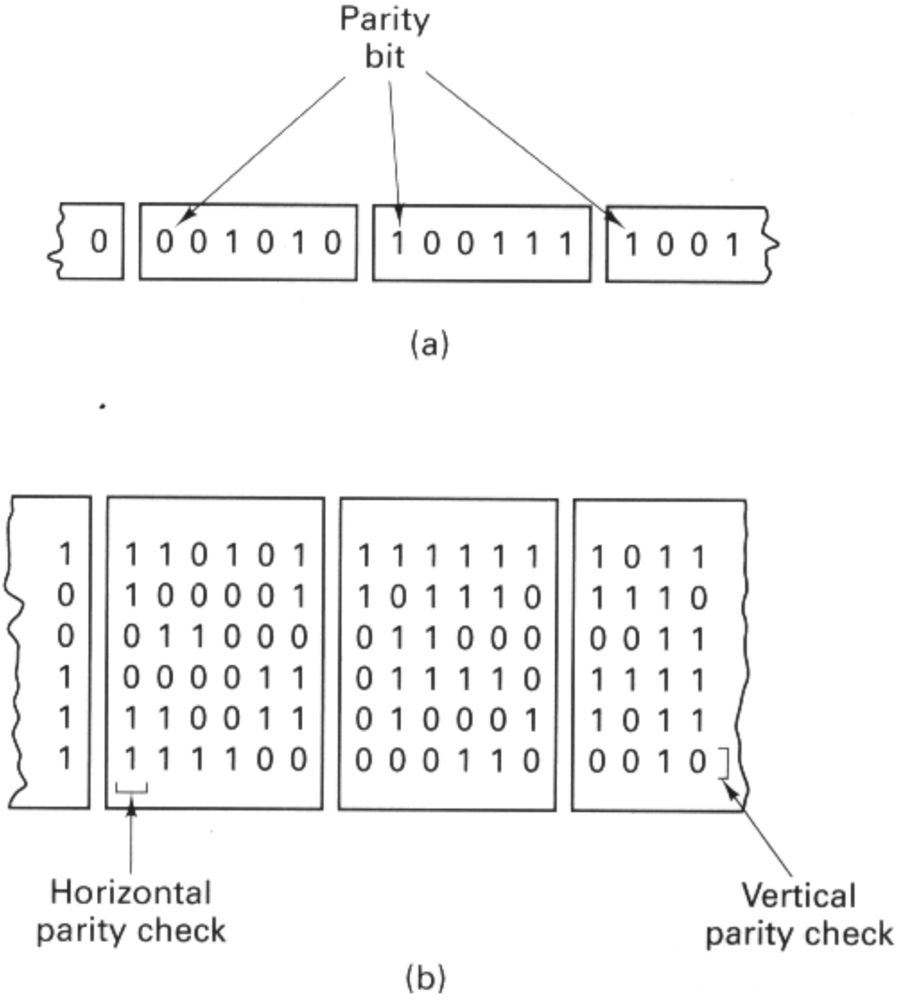


Figure 6.8 Parity checks for serial and parallel code structures. (a) Serial structure. (b) Parallel structure.

all bit errors are equally likely and occur independently, we can write the probability of j errors occurring in a block of n symbols as

$$P(j, n) = \binom{n}{j} p^j (1 - p)^{n-j} \tag{6.15}$$

where p is the probability that a *channel symbol* is received in error, and where

$$\binom{n}{j} = \frac{n!}{j!(n-j)!} \tag{6.16}$$

is the number of various ways in which j bits out of n may be in error. Thus, for a single-parity error-detection code, the probability of an undetected error P_{nd} with a block of n bits is computed as follows:

$$P_{nd} = \sum_{j=1}^{\substack{n/2 \text{ (for } n \text{ even)} \\ (n-1)/2 \text{ (for } n \text{ odd)}}} \binom{n}{2j} p^{2j} (1 - p)^{n-2j} \tag{6.17}$$

Example 6.1 Even-Parity Code

Configure a (4, 3) even-parity error-detection code such that the parity symbol appears as the leftmost symbol of the codeword. Which error patterns can the code detect? Compute the probability of an undetected message error, assuming that all symbol errors are independent events and that the probability of a channel symbol error is $p = 10^{-3}$.

Solution

Message	Parity	Codeword
000	0	0 000
100	1	1 100
010	1	1 010
110	0	0 110
001	1	1 001
101	0	0 101
011	0	0 011
111	1	1 111

parity message

The code is capable of detecting all single- and triple-error patterns. The probability of an undetected error is equal to the probability that two or four errors occur anywhere in a codeword.

$$\begin{aligned} P_{nd} &= \binom{4}{2} p^2 (1 - p)^2 + \binom{4}{4} p^4 \\ &= 6p^2(1 - p)^2 + p^4 \\ &= 6p^2 - 12p^3 + 7p^4 \\ &= 6(10^{-3})^2 - 12(10^{-3})^3 + 7(10^{-3})^4 \approx 6 \times 10^{-6} \end{aligned}$$

6.3.3.2 Rectangular Code

A *rectangular code*, also called a *product code*, can be thought of as a parallel code structure, depicted in Figure 6.8b. First we form a rectangle of message bits comprising M rows and N columns; then, a horizontal parity check is appended to each row and a vertical parity check is appended to each column, resulting in an augmented array of dimensions $(M + 1) \times (N + 1)$. The rate of the rectangular code k/n can then be written as

$$\frac{k}{n} = \frac{MN}{(M + 1)(N + 1)}$$

How much more powerful is the rectangular code than the single-parity code, which is only capable of error detection? Notice that any single bit error will cause a parity check failure in one of the array columns *and* in one of the array rows. Therefore, the rectangular code can correct any single error pattern since such an error is uniquely located at the intersection of the error-detecting row and the error-detecting column. For the example shown in Figure 6.8b, the array dimensions are $M = N = 5$; therefore, the figure depicts a (36, 25) code that can correct a single error located anywhere in the 36-bit positions. For such an error-correcting block code, we compute the probability that the decoded block has an uncorrected error by accounting for all the ways in which a *message error* can be made. Starting with the probability of j errors in a block of n symbols, expressed in Equation (6.15), we can write the probability of a message error, also called a *block error* or *word error*, for a code that can correct all t and fewer error patterns as

$$P_M = \sum_{j=t+1}^{n} \binom{n}{j} p^j (1 - p)^{n-j} \tag{6.18}$$

where p is the probability that a *channel symbol* is received in error. For the example in Figure 6.8b, the code can correct all single error patterns ($t = 1$) within the rectangular block of $n = 36$ bits. Hence, the summation in Equation (6.18) starts with $j = 2$:

$$P_M = \sum_{j=2}^{36} \binom{36}{j} p^j (1 - p)^{36-j}$$

When p is reasonably small, the first term in the summation is the dominant one; Therefore, for this (36, 25) rectangular code example, we can write

$$P_M \approx \binom{36}{2} p^2 (1 - p)^{34}$$

The *bit error probability* P_B depends on the particular code and decoder. An approximation for P_B is given in Section 6.5.3.

6.3.4 Why Use Error-Correction Coding?

Error-correction coding can be regarded as a vehicle for effecting various system trade-offs. Figure 6.9 compares two curves depicting bit-error performance versus

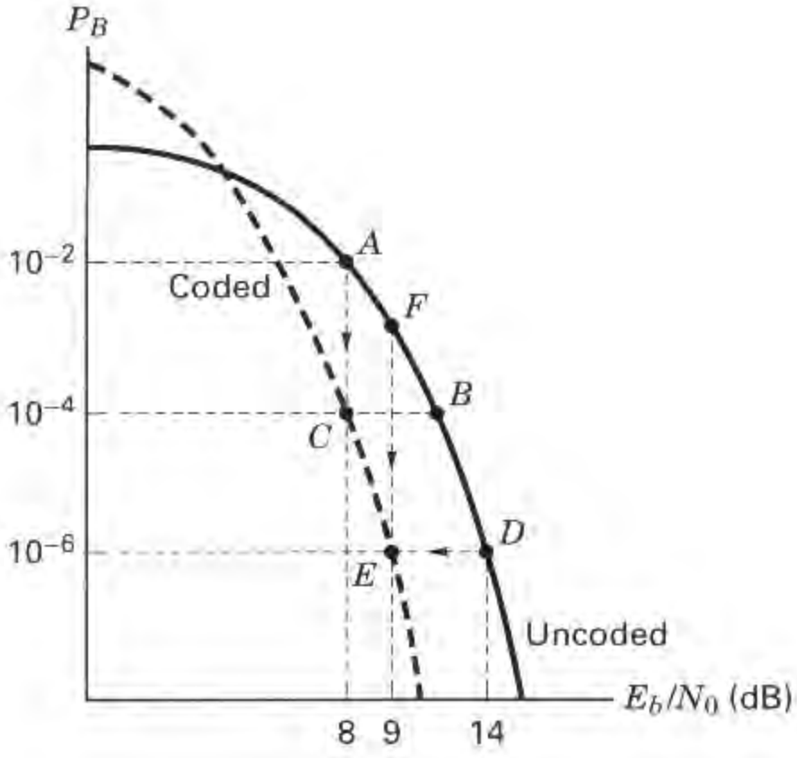


Figure 6.9 Comparison of typical coded versus uncoded error performance.

E_b/N_0. One curve represents a typical modulation scheme without coding. The second curve represents the same modulation with coding. Demonstrated below are four benefits or trade-offs that can be achieved with the use of channel coding.

6.3.4.1 Trade-Off 1: Error Performance versus Bandwidth

Imagine that a simple, inexpensive voice communication system has just been developed and delivered to a customer. The system does not use error-correction coding. Consider that the operating point of the system can be depicted by point A in Figure 6.9 ($E_b/N_0 = 8$ dB, and $P_B = 10^{-2}$). After a few trials, there are complaints about the voice quality; the customer suggests that the bit-error probability should be lowered to 10^{-4}. The usual way of obtaining better error performance in such a system would be by effecting an operating point movement from point A to, say, point B in Figure 6.9. However, suppose that the E_b/N_0 of 8 dB is the most that is available in this system. The figure suggests that one possible trade-off is to move the operating point from point A to point C. That is, "walking" down the vertical line to point C on the coded curve can provide the customer with improved error performance. What does it cost? Aside from the new components (encoder and decoder) needed, the price is more transmission bandwidth. Error-correction coding needs redundancy. If we assume that the system is a real-time communication system (such that the message may not be delayed), the addition of redundant bits dictates a faster rate of transmission, which of course means more bandwidth.

6.3.4.2 Trade-Off 2: Power versus Bandwidth

Consider that a system without coding, operating at point D in Figure 6.9 ($E_b/N_0 = 14$ dB, and $P_B = 10^{-6}$), has been delivered to a customer. The customer has no complaints about the quality of the data, but the equipment is having some

reliability problems as a result of providing an E_b/N_0 of 14 dB. In other words, the equipment keeps breaking down. If the requirement on E_b/N_0 or power could be reduced, the reliability difficulties might also be reduced. Figure 6.9 suggests a trade-off by moving the operating point from point D to point E. That is, if error-correction coding is introduced, a reduction in the required E_b/N_0 can be achieved. Thus, the trade-off is one in which the same quality of data is achieved, but the coding allows for a reduction in power or E_b/N_0. What is the cost? The same as before—more bandwidth.

Notice that for *non-real-time* communication systems, error-correction coding can be used with a somewhat different trade-off. It is possible to obtain improved bit-error probability or reduced power (similar to trade-off 1 or 2 above) by paying the price of *delay* instead of bandwidth.

6.3.4.3 Coding Gain

The trade-off example described in the previous section has allowed a reduction in E_b/N_0 from 14 dB to 9 dB, while maintaining the same error performance. In the context of this example and Figure 6.9, we now define *coding gain*. For a *given bit-error probability*, coding gain is defined as the "relief" or reduction in E_b/N_0 that can be realized through the use of the code. Coding gain G is generally expressed in dB, such as

$$G(\text{dB}) = \left(\frac{E_b}{N_0}\right)_u (\text{dB}) - \left(\frac{E_b}{N_0}\right)_c (\text{dB}) \tag{6.19}$$

where $(E_b/N_0)_u$ and $(E_b/N_0)_c$ represent the required E_b/N_0, uncoded and coded, respectively.

6.3.4.4 Trade-Off 3: Data Rate versus Bandwidth

Consider that a system without coding, operating at point D in Figure 6.9 ($E_b/N_0 = 14$ dB, and $P_B = 10^{-6}$) has been developed. Assume that there is no problem with the data quality and no particular need to reduce power. However, in this example, suppose that the customer's data rate requirement increases. Recall the relationship in Equation (5.20b):

$$\frac{E_b}{N_0} = \frac{P_r}{N_0}\left(\frac{1}{R}\right)$$

If we do nothing to the system except increase the data rate R, the above expression shows that the received E_b/N_0 would decrease, and in Figure 6.9, the operating point would move upwards from point D to, let us say, some point F. Now, envision "walking" down the vertical line to point E on the curve that represents coded modulation. Increasing the data rate has degraded the quality of the data. But, the use of error-correction coding brings back the same quality at the same power level (P_r/N_0). The E_b/N_0 is reduced, but the code facilitates getting the same error probability with a lower E_b/N_0. What price do we pay for getting this higher data rate or greater capacity? The same as before—increased bandwidth.

6.3.4.5 Trade-Off 4: Capacity versus Bandwidth

Trade-off 4 is similar to trade-off 3 because both achieve increased capacity. A spread-spectrum multiple access technique, called code-division multiple access (CDMA) and described in Chapter 12, is one of the schemes used in cellular telephony. In CDMA, where users simultaneously share the same spectrum, each user is an interferer to each of the other users in the same cell or nearby cells. Hence, the capacity (maximum number of users) per cell is inversely proportional to E_b/N_0. (See Section 12.8.) In this application, a lowered E_b/N_0 results in a raised capacity; the code achieves a reduction in each user's power, which in turn allows for an increase in the number of users. Again, the cost is more bandwidth. But, in this case, the signal-bandwidth expansion due to the error-correcting code is small compared with the more significant spread-spectrum bandwidth expansion, and thus, there is no impact on the transmission bandwidth.

In each of the above trade-off examples, a "traditional" code involving redundant bits and faster signaling (for a real-time communication system) has been assumed; hence, in each case, the cost was expanded bandwidth. However, there exists an error-correcting technique, called *trellis-coded modulation*, that does not require faster signaling or expanded bandwidth for real-time systems. (This technique is described in Section 9.10.)

Example 6.2 Coded versus Uncoded Performance

Compare the message error probability for a communications link with and without the use of error-correction coding. Assume that the uncoded transmission characteristics are: BPSK modulation, Gaussian noise, $P_r/N_0 = 43{,}776$, data rate $R = 4800$ bits/s. For the coded case, also assume the use of a (15, 11) error-correcting code that is capable of correcting any single-error pattern within a block of 15 bits. Consider that the demodulator makes hard decisions and thus feeds the demodulated code bits directly to the decoder, which in turn outputs an estimate of the original message.

Solution

Following Equation (4.79), let $p_u = Q\sqrt{2E_b/N_0}$ and $p_c = Q\sqrt{2E_c/N_0}$ be the uncoded and coded channel symbol error probabilities, respectively, where E_b/N_0 is the bit energy per noise spectral density and E_c/N_0 is the code-bit energy per noise spectral density.

Without coding

$$\frac{E_b}{N_0} = \frac{P_r}{N_0}\left(\frac{1}{R}\right) = 9.12\,(9.6\text{ dB})$$

and

$$p_u = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) = Q(\sqrt{18.24}) = 1.02 \times 10^{-5} \tag{6.20}$$

where the following approximation of $Q(x)$ from Equation (3.44) was used:

$$Q(x) \approx \frac{1}{x\sqrt{2\pi}} \exp\left(\frac{-x^2}{2}\right) \qquad \text{for } x > 3$$

The probability that the uncoded message block P_M^u will be received in error is 1 minus the product of the probabilities that each bit will be detected correctly. Thus,

$$\begin{aligned} P_M^u &= 1 - (1 - p_u)^k \\ &= 1 - \underbrace{(1 - p_u)^{11}}_{\substack{\text{probability that all} \\ \text{11 bits in uncoded} \\ \text{block are correct}}} \qquad = \underbrace{1.12 \times 10^{-4}}_{\substack{\text{probability that at} \\ \text{least 1 bit out of} \\ \text{11 is in error}}} \end{aligned} \tag{6.21}$$

With coding

Assuming a *real-time* communication system such that delay is unacceptable, the channel-symbol rate or code-bit rate R_c is 15/11 times the data bit rate:

$$R_c = 4800 \times \tfrac{15}{11} \approx 6545 \text{ bps}$$

and

$$\frac{E_c}{N_0} = \frac{P_r}{N_0}\left(\frac{1}{R_c}\right) = 6.69 \,(8.3 \text{ dB})$$

The E_c/N_0 for each code bit is less than that for the data bit in the uncoded case because the channel-bit rate has increased, but the transmitter power is assumed to be fixed:

$$p_c = Q\left(\sqrt{\frac{2E_c}{N_0}}\right) = Q(\sqrt{13.38} = 1.36 \times 10^{-4} \tag{6.22}$$

It can be seen by comparing the results of Equation 6.20 with those of Equation 6.22 that because redundancy was added, the channel bit-error probability has degraded. More bits must be detected during the same time interval and with the same available power: the performance improvement due to the coding *is not yet apparent.* We now compute the coded message error rate P_M^c, using Equation 6.18:

$$P_M^c = \sum_{j=2}^{n=15} \binom{15}{j} (p_c)^j (1 - p_c)^{15-j}$$

The summation is started with $j = 2$, since the code corrects all single errors within a block of $n = 15$ bits. An approximation is obtained by using only the first term of the summation. For p_c, we use the value calculated in Equation 6.22:

$$P_M^c \approx \binom{15}{2} (p_c)^2 (1 - p_c)^{13} = 1.94 \times 10^{-6} \tag{6.23}$$

By comparing the results of Equation (6.21) with (6.23), we can see that the probability of message error has improved by a factor of 58 due to the error-correcting code used in this example. This example illustrates the typical behavior of all such real-time communication systems using error-correction coding. Added redundancy means faster signaling, less energy per channel symbol, and more errors out of the demodulator. The benefits arise because the behavior of the decoder will (at reasonable values of E_b/N_0) more than compensate for the poor performance of the demodulator.

6.3.4.6 Code Performance at Low Values of E_b/N_0

The reader is urged to solve Problem 6.5, which is similar to Example 6.2. In part (a) of Problem 6.5, where an E_b/N_0 of 14 dB is given, the result is a message-error performance improvement through the use of coding. However, in part (b) where the E_b/N_0 has been reduced to 10 dB, coding provides no improvement; in fact, there is a degradation. One might ask, Why does part (b) manifest a degradation? After all, the same procedure is used for applying the code in both parts of the problem. The answer can be seen in the coded-versus-uncoded pictorial shown in Figure 6.9. Even though Problem 6.5 deals with message-error probability, and Figure 6.9 displays bit-error probability, the following explanation still applies. In all such plots, there is a crossover between the curves (usually at some low value of E_b/N_0). The reason for such crossover (threshold) is that every code system has some fixed error-correcting capability. If there are more errors within a block than the code is capable of correcting, the system will perform poorly. Imagine that E_b/N_0 is continually reduced. What happens at the output of the demodulator? It makes more and more errors. Therefore, such a continual decrease in E_b/N_0 must eventually cause some threshold to be reached where the decoder becomes overwhelmed with errors. When that threshold is crossed, we can interpret the degraded performance as being caused by the redundant bits consuming energy but giving back nothing beneficial in return. Does it strike the reader as a paradox that operating in a region (low values of E_b/N_0), where one would best like to see an error-performance improvement, is where the code makes things worse? There is, however, a class of powerful codes called *turbo codes* that provide error-performance improvements at low values of E_b/N_0; the crossover point is lower for turbo codes compared with conventional codes. (These are treated in Section 8.4.)

6.4 LINEAR BLOCK CODES

Linear block codes (such as the one described in Example 6.2) are a class of parity-check codes that can be characterized by the (n, k) notation described earlier. The encoder transforms a block of k message digits (a message vector) into a longer block of n codeword digits (a code vector) constructed from a given alphabet of elements. When the alphabet consists of two elements (0 and 1), the code is a binary code comprising binary digits (bits). Our discussion of linear block codes is restricted to binary codes unless otherwise noted.

The k-bit messages form 2^k distinct message sequences, referred to as *k-tuples* (sequences of k digits). The n-bit blocks can form as many as 2^n distinct sequences, referred to as *n-tuples*. The encoding procedure assigns to each of the 2^k message k-tuples *one* of the 2^n n-tuples. A block code represents a one-to-one assignment, whereby the 2^k message k-tuples are *uniquely* mapped into a new set of 2^k codeword n-tuples; the mapping can be accomplished via a look-up table. For *linear codes*, the mapping transformation is, of course *linear*.

6.4.1 Vector Spaces

The set of all binary n-tuples, V_n, is called a *vector space* over the binary field of two elements (0 and 1). The binary field has two operations, addition and multiplication, such that the results of all operations are in the same set of two elements. The arithmetic operations of addition and multiplication are defined by the conventions of the algebraic field [4]. For example, in a binary field, the rules of addition and multiplication are as follows:

Addition	Multiplication
$0 \oplus 0 = 0$	$0 \cdot 0 = 0$
$0 \oplus 1 = 1$	$0 \cdot 1 = 0$
$1 \oplus 0 = 1$	$1 \cdot 0 = 0$
$1 \oplus 1 = 0$	$1 \cdot 1 = 1$

The addition operation, designated with the symbol $\oplus$, is the same modulo-2 operation described in Section 2.9.3. The summation of binary n-tuples always entails modulo-2 addition. However, for notational simplicity the ordinary + sign will often be used.

6.4.2 Vector Subspaces

A subset S of the vector space V_n is called a *subspace* if the following two conditions are met:

1. The all-zeros vector is in S.
2. The sum of any two vectors in S is also in S (known as the *closure property*).

These properties are fundamental for the algebraic characterization of *linear block codes.* Suppose that $\mathbf{V}_i$ and $\mathbf{V}_j$ are two codewords (or code vectors) in an (n, k) binary block code. The code is said to be *linear* if, and only if $(\mathbf{V}_i \oplus \mathbf{V}_j)$ is also a code vector. A linear block code, then, is one in which vectors outside the subspace cannot be created by the addition of legitimate codewords (members of the subspace).

For example, the vector space V_4 is totally populated by the following 2^4 = sixteen 4-tuples:

0000	0001	0010	0011	0100	0101	0110	0111
1000	1001	1010	1011	1100	1101	1110	1111

An example of a subset of V_4 that forms a subspace is

0000 0101 1010 1111

It is easy to verify that the addition of any two vectors in the subspace can only yield one of the other members of the subspace. A set of 2^k n-tuples is called a *linear block code* if, and only if, it is a subspace of the vector space V_n of all n-tuples. Figure 6.10 illustrates, with a simple geometric analogy, the structure behind linear

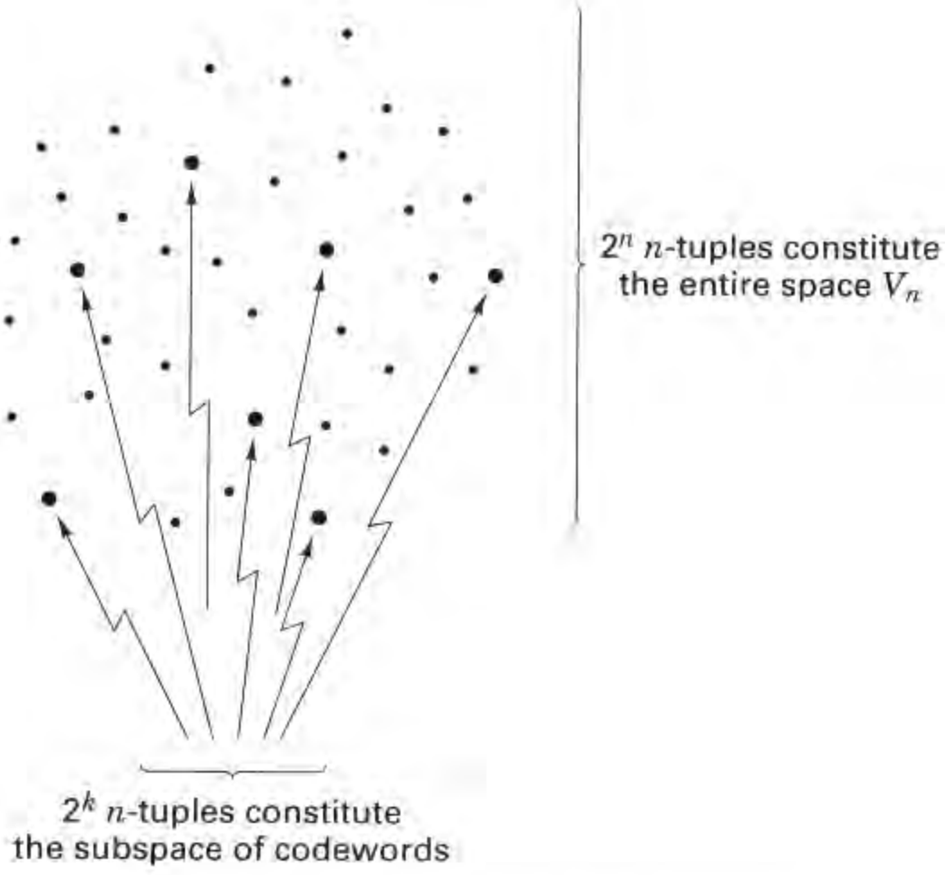


Figure 6.10 Linear block-code structure.

block codes. We can imagine the vector space V_n comprising 2^n n-tuples. Within this vector space there exists a subset of 2^k n-tuples making up a subspace. These 2^k vectors or points, shown "sprinkled" among the more numerous 2^n points, represent the legitimate or allowable codeword assignments. A message is encoded into one of the 2^k allowable code vectors and then transmitted. Because of noise in the channel, a perturbed version of the codeword (one of the other 2^n vectors in the n-tuple space) may be received. If the perturbed vector is not too unlike (not too distant from) the valid codeword, the decoder can decode the message correctly. The basic goals in choosing a particular code, similar to the goals in selecting a set of modulation waveforms, can be stated in the context of Figure 6.10 as follows:

1. We strive for coding efficiency by packing the V_n space with as many codewords as possible. This is tantamount to saying that we only want to expend a *small amount of redundancy* (excess bandwidth).
2. We want the codewords to be as *far apart from one another* as possible, so that even if the vectors experience some corruption during transmission, they may still be correctly decoded, with a high probability.

6.4.3 A (6, 3) Linear Block Code Example

Examine the following coding assignment that describes a (6, 3) code. There are $2^k = 2^3 = 8$ message vectors, and therefore eight codewords. There are $2^k = 2^6 =$ sixty-four 6-tuples in the V_6 vector space:

It is easy to check that the eight codewords shown in Table 6.1 form a subspace of V_6 (the all-zeros vector is present, and the sum of any two codewords yields another codeword member of the subspace). Therefore, these codewords represent a *linear block code*, as defined in Section 6.4.2. A natural question to ask is How was the codeword-to-message assignment chosen for this (6, 3) code? A unique assignment for a particular (n, k) code does not exist; however, neither is there complete freedom of choice. In Section 6.6.3, we examine the requirements and constraints that drive a code design.

TABLE 6.1 Assignment of Codewords to Messages

Message vector	Codeword
000	000000
100	110100
010	011010
110	101110
001	101001
101	011101
011	110011
111	000111

6.4.4 Generator Matrix

If k is large, a *table look-up* implementation of the encoder becomes prohibitive. For a (127, 92) code there are 2^{92} or approximately 5×10^{27} code vectors. If the encoding procedure consists of a simple look-up table, imagine the size of the memory necessary to contain such a large number of codewords. Fortunately, it is possible to reduce complexity by generating the required codewords as needed, instead of storing them.

Since a set of codewords that forms a linear block code is a k-dimensional subspace of the n-dimensional binary vector space ($k < n$), it is always possible to find a set of n-tuples, fewer than 2^k, that can generate all the 2^k codewords of the subspace. The generating set of vectors is said to *span* the subspace. The smallest *linearly independent* set that spans the subspace is called a *basis* of the subspace, and the number of vectors in this basis set is the dimension of the subspace. Any basis set of k linearly independent n-tuples $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_k$ can be used to generate the required linear block code vectors, since each code vector is a linear combination of $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_k$. That is, each of the set of 2^k codewords $\{\mathbf{U}\}$ can be described by

$$\mathbf{U} = m_1\mathbf{V}_1 + m_2\mathbf{V}_2 + \cdots + m_k\mathbf{V}_k$$

where $m_i = (0 \text{ or } 1)$ are the message digits and $i = 1, \ldots, k$.

In general, we can define a *generator matrix* by the following $k \times n$ array:

$$\mathbf{G} = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \vdots \\ \mathbf{V}_k \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & & & \\ v_{k1} & v_{k2} & \cdots & v_{kn} \end{bmatrix} \tag{6.24}$$

Code vectors, by convention, are usually designated as row vectors. Thus, the message **m**, a sequence of k message bits, is shown below as a row vector ($1 \times k$ matrix having one row and k columns):

$$\mathbf{m} = m_1, m_2, \ldots, m_k$$

The generation of the codeword **U** is written in matrix notation as the product of **m** and **G**, and we write

$$\mathbf{U} = \mathbf{mG} \tag{6.25}$$

where, in general, the matrix multiplication $\mathbf{C} = \mathbf{AB}$ is performed in the usual way by using the rule

$$c_{ij} = \sum_{k}^{n} a_{ik} b_{kj} \qquad i = 1, \ldots, l \quad j = 1, \ldots, m$$

where **A** is an $l \times n$ matrix, **B** is an $n \times m$ matrix, and the result **C** is an $l \times m$ matrix. For the example introduced in the preceding section, we can fashion a generator matrix as

$$\mathbf{G} = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{6.26}$$

where $\mathbf{V}_1$, $\mathbf{V}_2$, and $\mathbf{V}_3$ are three *linearly independent vectors* (a subset of the eight code vectors) that can generate all the code vectors. Notice that the sum of any two generating vectors does not yield any of the other generating vectors (opposite of closure). Let us generate the codeword $\mathbf{U}_4$ for the fourth message vector 1 1 0 in Table 6.1, using the generator matrix of Equation (6.26):

$$\mathbf{U}_4 = [1 \quad 1 \quad 0] \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = 1 \cdot \mathbf{V}_1 + 1 \cdot \mathbf{V}_2 + 0 \cdot \mathbf{V}_3$$

$$= 1\ 1\ 0\ 1\ 0\ 0 + 0\ 1\ 1\ 0\ 1\ 0 + 0\ 0\ 0\ 0\ 0\ 0$$

$$= 1\ 0\ 1\ 1\ 1\ 0 \quad \text{(codeword for the message vector } 1\ 1\ 0\text{)}$$

Thus, the code vector corresponding to a message vector is a linear combination of the rows of **G**. Since the code is totally defined by **G**, the encoder need only store the k rows of G instead of the total 2^k vectors of the code. For this example, notice that the generator array of dimension 3×6 in Equation (6.26) replaces the original codeword array of dimension 8×6 in Table 6.1, representing a reduction in system complexity.

6.4.5 Systematic Linear Block Codes

A systematic (n, k) linear block code is a mapping from a k-dimensional message vector to an n-dimensional codeword in such a way that part of the sequence generated coincides with the k message digits. The remaining $(n - k)$ digits are parity digits. A systematic linear block code will have a generator matrix of the form

$$\mathbf{G} = \left[\begin{array}{c:c} \mathbf{P} & \mathbf{I}_k \end{array}\right]$$

$$= \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,(n-k)} & 1 & 0 & \cdots & 0 \\ p_{21} & p_{22} & \cdots & p_{2,(n-k)} & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & \vdots & \\ p_{k1} & p_{k2} & \cdots & p_{k,(n-k)} & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{6.27}$$

where $\mathbf{P}$ is the parity array portion of the generator matrix p_{ij} = (0 or 1), and $\mathbf{I}_k$ is the $k \times k$ identity matrix (ones on the main diagonal and zeros elsewhere). Notice that with this systematic generator, the encoding complexity is further reduced since it is not necessary to store the identity matrix portion of the array. By combining Equations (6.26) and (6.27), each codeword is expressed as

$$u_1, u_2, \ldots, u_n = [m_1, m_2, \ldots, m_k] \times \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1,(n-k)} & 1 & 0 & \cdots & 0 \\ p_{21} & p_{22} & \cdots & p_{2,(n-k)} & 0 & 1 & \cdots & 0 \\ \vdots & & & & & & & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{k,(n-k)} & 0 & 0 & \cdots & 1 \end{bmatrix}$$

where

$$\begin{aligned} u_i &= m_1 p_{1i} + m_2 p_{2i} + \cdots + m_k p_{ki} && \text{for } i = 1, \ldots, (n-k) \\ &= m_{i-n+k} && \text{for } i = (n-k+1), \ldots, n \end{aligned}$$

Given the message k-tuple

$$\mathbf{m} = m_1, m_2, \ldots, m_k$$

and the general code vector n-tuple

$$\mathbf{U} = u_1, u_2, \ldots, u_n$$

the systematic code vector can be expressed as

$$\mathbf{U} = \underbrace{p_1, p_2, \ldots, p_{n-k}}_{\text{parity bits}}, \underbrace{m_1, m_2, \ldots, m_k}_{\text{message bits}} \tag{6.28}$$

where

$$
\begin{aligned}
p_1 &= m_1 p_{11} + m_2 p_{21} + \cdots + m_k p_{k1} \\
p_2 &= m_1 p_{12} + m_2 p_{22} + \cdots + m_k p_{k2} \\
p_{n-k} &= m_1 p_{1,(n-k)} + m_2 p_{2,(n-k)} + \cdots + m_k p_{k,(n-k)}
\end{aligned} \tag{6.29}
$$

Systematic codewords are sometimes written so that the message bits occupy the left-hand portion of the codeword and the parity bits occupy the right-hand portion. This reordering has no effect on the error detection or error correction properties of the code, and will not be considered further.

For the (6, 3) code example in Section 6.4.3, the codewords are described as follows:

$$
\mathbf{U} = [m_1, m_2, m_3] \left[\begin{array}{ccc:ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] \tag{6.30}
$$

$$
\underbrace{\qquad\qquad}_{\mathbf{P}} \quad \underbrace{\qquad\qquad}_{\mathbf{I}_3}
$$

$$
= \underbrace{m_1 + m_3}_{u_1},\ \underbrace{m_1 + m_2}_{u_2},\ \underbrace{m_2 + m_3}_{u_3},\ \underbrace{m_1}_{u_4},\ \underbrace{m_2}_{u_5},\ \underbrace{m_3}_{u_6} \tag{6.31}
$$

Equation (6.31) provides some insight into the structure of linear block codes. We see that the redundant digits are produced in a variety of ways. The first parity bit is the sum of the first and third message bits; the second parity bit is the sum of the first and second message bits; and the third parity bit is the sum of the second and third message bits. Intuition tells us that such structure, compared with single-parity checks or simple digit-repeat procedures, may provide greater ability to detect and correct errors.

6.4.6 Parity-Check Matrix

Let us define a matrix **H**, called the *parity-check matrix*, that will enable us to decode the received vectors. For each $(k \times n)$ generator matrix **G**, there exists an $(n - k) \times n$ matrix **H**, such that the rows of **G** are orthogonal to the rows of **H**; that is, $\mathbf{G}\mathbf{H}^T = \mathbf{0}$, where $\mathbf{H}^T$ is the *transpose* of **H**, and **0** is a $k \times (n - k)$ all-zeros matrix. $\mathbf{H}^T$ is an $n \times (n - k)$ matrix whose rows are the columns of **H** and whose columns are the rows of **H**. To fulfill the orthogonality requirements for a systematic code, the components of the **H** matrix are written as

$$
\mathbf{H} = [\mathbf{I}_{n-k} \mid \mathbf{P}^T] \tag{6.32}
$$

Hence, the $\mathbf{H}^T$ matrix is written as

$$
\mathbf{H}^T = \begin{bmatrix} \mathbf{I}_{n-k} \\ \hdashline \mathbf{P} \end{bmatrix} \tag{6.33a}
$$

$$= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \\ p_{11} & p_{12} & \cdots & p_{1,(n-k)} \\ p_{21} & p_{22} & \cdots & p_{2,(n-k)} \\ \vdots & & & \\ p_{k1} & p_{k2} & \cdots & p_{k,(n-k)} \end{bmatrix} \tag{6.33b}$$

It is easy to verify that the product $\mathbf{UH}^T$ of each codeword **U,** generated by **G** and the $\mathbf{H}^T$ matrix, yields

$$\mathbf{UH}^T = p_1 + p_1,\ p_2 + p_2,\ \ldots,\ p_{n-k} + p_{n-k} = \mathbf{0}$$

where the parity bits $p_1, p_2, \ldots p_{n-k}$ are defined in Equation (6.29). Thus, once the *parity-check matrix* **H** is constructed to fulfill the foregoing orthogonality requirements, we can use it to test whether a received vector is a valid member of the codeword set. **U** is a codeword generated by matrix **G** if, and only if $\mathbf{UH}^T = \mathbf{0}$.

6.4.7 Syndrome Testing

Let $\mathbf{r} = r_1, r_2, \ldots, r_n$ be a received vector (one of 2^n n-tuples) resulting from the transmission of $\mathbf{U} = u_1, u_2, \ldots, u_n$ (one of 2^k n-tuples). We can therefore describe **r** as

$$\mathbf{r} = \mathbf{U} + \mathbf{e} \tag{6.34}$$

where $\mathbf{e} = e_1, e_2, \ldots, e_n$ is an error vector or error pattern introduced by the channel. There are a total of $2^n - 1$ potential nonzero error patterns in the space of 2^n n-tuples. The *syndrome* of **r** is defined as

$$\mathbf{S} = \mathbf{rH}^T \tag{6.35}$$

The syndrome is the result of a parity check performed on **r** to determine whether **r** is a valid member of the codeword set. If, in fact, **r** is a member, the syndrome **S** has a value **0**. If **r** contains detectable errors, the syndrome has some nonzero value. If **r** contains correctable errors, the syndrome (like the symptom of an illness) has some nonzero value that can earmark the particular error pattern. The decoder, depending upon whether it has been implemented to perform FEC or ARQ, will then take actions to locate the errors and correct them (FEC), or else it will request a retransmission (ARQ). Combining Equations (6.34) and (6.35), the syndrome of **r** is seen to be

$$\begin{aligned} \mathbf{S} &= (\mathbf{U} + \mathbf{e})\mathbf{H}^T \\ &= \mathbf{UH}^T + \mathbf{eH}^T \end{aligned} \tag{6.36}$$

However, $\mathbf{UH}^T = \mathbf{0}$ for all members of the codeword set. Therefore,

$$\mathbf{S} = \mathbf{eH}^T \tag{6.37}$$

The foregoing development, starting with Equation (6.34) and terminating with Equation (6.37), is evidence that the syndrome test, whether performed on either a corrupted code vector or on the error pattern that caused it, yields the same syndrome. An important property of linear block codes, fundamental to the decoding process, is that the mapping between correctable error patterns and syndromes is one to one.

It is interesting to note the following two required properties of the parity-check matrix.

1. No column of **H** can be all zeros, or else an error in the corresponding codeword position would not affect the syndrome and would be undetectable.
2. All columns of **H** must be unique. If two columns of **H** were identical, errors in these two corresponding codeword positions would be indistinguishable.

Example 6.3 Syndrome Test

Suppose that codeword $\mathbf{U} = 1\ 0\ 1\ 1\ 1\ 0$ from the example in Section 6.4.3 is transmitted and the vector $\mathbf{r} = 0\ 0\ 1\ 1\ 1\ 0$ is received; that is, the leftmost bit is received in error. Find the syndrome vector value $\mathbf{S} = \mathbf{r}\mathbf{H}^T$ and verify that it is equal to $\mathbf{e}\mathbf{H}^T$.

Solution

$$\mathbf{S} = \mathbf{r}\mathbf{H}^T$$

$$= [0\ 0\ 1\ 1\ 1\ 0] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= [1,\ 1+1,\ 1+1] = [1\ 0\ 0] \quad \text{(syndrome of corrupted code vector)}$$

Next, we verify that the syndrome of the corrupted code vector is the same as the syndrome of the error pattern that caused the error:

$$\mathbf{S} = \mathbf{e}\mathbf{H}^T = [1\ 0\ 0\ 0\ 0\ 0]\mathbf{H}^T = [1\ 0\ 0] \quad \text{(syndrome of error pattern)}$$

6.4.8 Error Correction

We have detected a single error and have shown that the syndrome test performed on either the corrupted codeword, or on the error pattern that caused it, yields the same syndrome. This should be a clue that we not only can detect the error, but since there is a one-to-one correspondence between correctable error patterns and syndromes, we can correct such error patterns. Let us arrange the 2^n n-tuples that represent possible received vectors in an array, called the *standard array,* such that the first row contains all the codewords, starting with the all-zeros codeword, and the first column contains all the correctable error patterns. Recall from the basic properties of linear codes (see Section 6.4.2) that the all-zeros vector must be a member of the codeword set. Each row, called a *coset,* consists of an error pattern

in the first column, called the *coset leader,* followed by the codewords perturbed by that error pattern. The standard array format for an (n, k) code is as follows:

$$\begin{array}{llllll} \mathbf{U}_1 & \mathbf{U}_2 & \cdots & \mathbf{U}_i & \cdots & \mathbf{U}_{2^k} \\ \mathbf{e}_2 & \mathbf{U}_2 + \mathbf{e}_2 & \cdots & \mathbf{U}_i + \mathbf{e}_2 & \cdots & \mathbf{U}_{2^k} + \mathbf{e}_2 \\ \mathbf{e}_3 & \mathbf{U}_2 + \mathbf{e}_3 & \cdots & \mathbf{U}_i + \mathbf{e}_3 & \cdots & \mathbf{U}_{2^k} + \mathbf{e}_3 \\ \vdots & \vdots & & \vdots & & \\ \mathbf{e}_j & \mathbf{U}_2 + \mathbf{e}_j & \cdots & \mathbf{U}_i + \mathbf{e}_j & \cdots & \mathbf{U}_{2^k} + \mathbf{e}_j \\ \vdots & \vdots & & \vdots & & \\ \mathbf{e}_{2^{n-k}} & \mathbf{U}_2 + \mathbf{e}_{2^{n-k}} & \cdots & \mathbf{U}_i + \mathbf{e}_{2^{n-k}} & \cdots & \mathbf{U}_{2^k} + \mathbf{e}_{2^{n-k}} \end{array} \tag{6.38}$$

Note that codeword $\mathbf{U}_1$, the all-zeros codeword, plays two roles. It is one of the codewords, and it can also be thought of as the error pattern $\mathbf{e}_1$—the pattern that represents no error, such that $\mathbf{r} = \mathbf{U}$. The array contains all 2^n n-tuples in the space V_n. Each n-tuple appears in *only one* location—none are missing, and none are replicated. Each coset consists of 2^k n-tuples. Therefore, there are $(2^n/2^k) = 2^{n-k}$ cosets.

The decoding algorithm calls for replacing a corrupted vector (any n-tuple excluding those in the first row) with a valid codeword from the top of the column containing the corrupted vector. Suppose that a codeword $\mathbf{U}_i$ $(i = 1, \ldots, 2^k)$ is transmitted over a noisy channel, resulting in a received (corrupted) vector $\mathbf{U}_i + \mathbf{e}_j$. If the error pattern $\mathbf{e}_j$ caused by the channel is a coset leader, where the index $j = 1, \ldots, 2^{n-k}$, the received vector will be decoded correctly into the transmitted codeword $\mathbf{U}_i$. If the error pattern is not a coset leader, then an erroneous decoding will result.

6.4.8.1 The Syndrome of a Coset

If $\mathbf{e}_j$ is the coset leader or error pattern of the jth coset, then $\mathbf{U}_i + \mathbf{e}_j$ is an n-tuple in this coset. The syndrome of this n-tuple can be written

$$\mathbf{S} = (\mathbf{U}_i + \mathbf{e}_j)\mathbf{H}^T = \mathbf{U}_i\mathbf{H}^T + \mathbf{e}_j\mathbf{H}^T$$

Since $\mathbf{U}_i$ is a code vector, $\mathbf{U}_i\mathbf{H}^T = \mathbf{0}$, and we can write, as in Equation (6.37),

$$\mathbf{S} = (\mathbf{U}_i + \mathbf{e}_j)\mathbf{H}^T = \mathbf{e}_j\mathbf{H}^T \tag{6.39}$$

The name *coset* is short for "a *set* of numbers having a *common* feature." What do the members of any given row (coset) have in common? From Equation (6.39) it is clear that each member of a coset has the *same syndrome.* The syndrome for each coset is different from that of any other coset in the code; it is the syndrome that is used to estimate the error pattern.

6.4.8.2 Error Correction Decoding

The procedure for error correction decoding proceeds as follows:

1. Calculate the syndrome of $\mathbf{r}$ using $\mathbf{S} = \mathbf{r}\mathbf{H}^T$.
2. Locate the coset leader (error pattern) $\mathbf{e}_j$, whose syndrome equals $\mathbf{r}\mathbf{H}^T$.

3. This error pattern is assumed to be the corruption caused by the channel.
4. The corrected received vector, or codeword, is identified as $\mathbf{U} = \mathbf{r} + \mathbf{e}_j$. We can say that we retrieve the valid codeword by subtracting out the identified error; in modulo-2 arithmetic, the operation of subtraction is identical to that of addition.

6.4.8.3 Locating the Error Pattern

Returning to the example of Section 6.4.3, we arrange the 2^6 = sixty-four 6-tuples in a standard array as shown in Figure 6.11. The valid codewords are the eight vectors in the first row, and the *correctable error patterns* are the seven nonzero *coset leaders* in the first column. Notice that all 1-bit error patterns are correctable. Also notice that after exhausting all 1-bit error patterns, there remains some error-correcting capability since we have not yet accounted for all sixty-four 6-tuples. There is one unassigned coset leader; therefore, there remains the capability of correcting one additional error pattern. We have the flexibility of choosing this error pattern to be any of the n-tuples in the remaining coset. In Figure 6.11 this final correctable error pattern is chosen, somewhat arbitrarily, to be the 2-bit error pattern 0 1 0 0 0 1. Decoding will be correct if, and only if, the error pattern caused by the channel is one of the coset leaders.

000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011001	110111	000011
001000	111100	010010	100110	100001	010101	111011	001111
010000	100100	001010	111110	111001	001101	100011	010111
100000	010100	111010	001110	001001	111101	010011	100111
010001	100101	001011	111111	111000	001100	100010	010110

Figure 6.11 Example of a standard array for a (6, 3) code.

We now determine the syndrome corresponding to each of the correctable error sequences by computing $\mathbf{e}_j\mathbf{H}^T$ for each coset leader, as follows:

$$\mathbf{S} = \mathbf{e}_j \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

The results are listed in Table 6.2. Since each syndrome in the table is unique, the decoder can identify the error pattern **e** to which it corresponds.

TABLE 6.2 Syndrome Look-Up Table

Error pattern	Syndrome
0 0 0 0 0 0	0 0 0
0 0 0 0 0 1	1 0 1
0 0 0 0 1 0	0 1 1
0 0 0 1 0 0	1 1 0
0 0 1 0 0 0	0 0 1
0 1 0 0 0 0	0 1 0
1 0 0 0 0 0	1 0 0
0 1 0 0 0 1	1 1 1

6.4.8.4 Error Correction Example

As outlined in Section 6.4.8.2 we receive the vector **r** and calculate its syndrome using $\mathbf{S} = \mathbf{r}\mathbf{H}^T$. We then use the syndrome look-up table (Table 6.2), developed in the preceding section, to find the corresponding error pattern. This error pattern is an estimate of the error, and we denote it $\hat{\mathbf{e}}$. The decoder then adds $\hat{\mathbf{e}}$ to **r** to obtain an estimate of the transmitted codeword $\hat{\mathbf{U}}$:

$$\hat{\mathbf{U}} = \mathbf{r} + \hat{\mathbf{e}} = (\mathbf{U} + \mathbf{e}) + \hat{\mathbf{e}} = \mathbf{U} + (\mathbf{e} + \hat{\mathbf{e}}) \tag{6.40}$$

If the estimated error pattern is the same as the actual error pattern, that is, if $\hat{\mathbf{e}} = \mathbf{e}$, then the estimate $\hat{\mathbf{U}}$ is equal to the transmitted codeword **U**. On the other hand, if the error estimate is incorrect, the decoder will estimate a codeword that was not transmitted, and we have an *undetectable decoding error.*

Example 6.4 Error Correction

Assume that codeword $\mathbf{U} = 1\ 0\ 1\ 1\ 1\ 0$, from the Section 6.4.3 example, is transmitted, and the vector $\mathbf{r} = 0\ 0\ 1\ 1\ 1\ 0$ is received. Show how a decoder, using the Table 6.2 syndrome look-up table, can correct the error.

Solution

The syndrome of **r** is computed:

$$\mathbf{S} = [0\ 0\ 1\ 1\ 1\ 0]\mathbf{H}^T = [1\ 0\ 0]$$

Using Table 6.2, the error pattern corresponding to the syndrome above is estimated to be

$$\hat{\mathbf{e}} = 1\ 0\ 0\ 0\ 0\ 0$$

The corrected vector is then estimated by

$$\begin{aligned}\hat{\mathbf{U}} &= \mathbf{r} + \hat{\mathbf{e}} \\ &= 0\ 0\ 1\ 1\ 1\ 0 + 1\ 0\ 0\ 0\ 0\ 0 \\ &= 1\ 0\ 1\ 1\ 1\ 0\end{aligned}$$

Since the estimated error pattern is the actual error pattern in this example, the error correction procedure yields $\hat{\mathbf{U}} = \mathbf{U}$.

Notice that the process of decoding a corrupted codeword by first detecting and then correcting an error can be compared to a familiar medical analogy. A patient (potentially corrupted codeword) enters a medical facility (decoder). The examining physician performs diagnostic testing (multiplies by $\mathbf{H}^T$) in order to find a symptom (syndrome). Imagine that the physician finds characteristic spots on the patient's x-rays. An experienced physician would immediately recognize the correspondence between the symptom and the disease (error pattern) tuberculosis. A novice physician might have to refer to a medical handbook (Table 6.2) to associate the symptom or syndrome with the disease or error pattern. The final step is to provide the proper medication that removes the disease, as seen in Equation (6.40). In the context of binary codes and the medical analogy, Equation (6.40) reveals an unusual type of medicine practiced here. The patient is cured by reapplying the original disease.

6.4.9 Decoder Implementation

When the code is short as in the case of the (6, 3) code described in the previous sections, the decoder can be implemented with simple circuitry. Consider the steps that the decoder must take: (1) calculate the syndrome, (2) locate the error pattern, and (3) perform modulo-2 addition of the error pattern and the received vector (which removes the error). In Example 6.4, we started with a corrupted vector and saw how these steps yielded the corrected codeword. Now, consider the circuit in Figure 6.12, made up of exclusive-OR gates and AND gates that can accomplish the same result for any *single-error pattern* in the (6, 3) code. From Table 6.2 and Equation 6.39, we can write an expression for each of the syndrome digits in terms of the received codeword digits as

$$\mathbf{S} = \mathbf{r}\mathbf{H}^T$$

$$\mathbf{S} = [r_1\ r_2\ r_3\ r_4\ r_5\ r_6] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

and

$$\begin{aligned} s_1 &= r_1 + r_4 + r_6 \\ s_2 &= r_2 + r_4 + r_5 \\ s_3 &= r_3 + r_5 + r_6 \end{aligned}$$

We use these syndrome expressions for wiring up the circuit in Figure 6.12. The exclusive-OR gate is the same operation as modulo-2 arithmetic and hence uses the same symbol. A small circle at the termination of any line entering the AND gate indicates the logic COMPLEMENT of the signal.

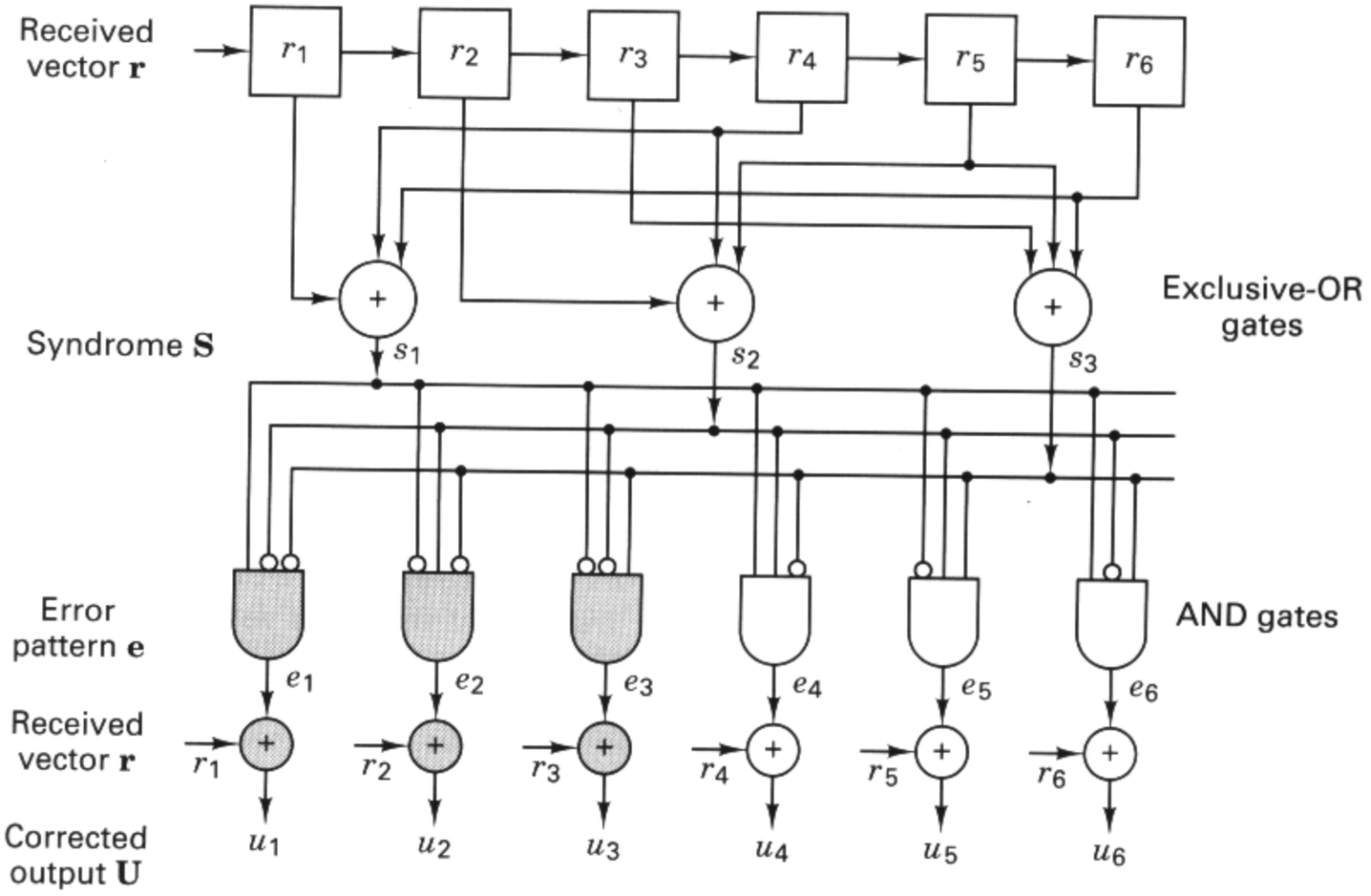


Figure 6.12 Implementation of the (6, 3) decoder.

The corrupted signal enters the decoder at two places simultaneously. At the upper part of the circuit, the syndrome is computed, and at the lower part, that syndrome is transformed to its corresponding error pattern. The error is removed by adding it back to the received vector yielding the corrected codeword.

Note that, for tutorial reasons, Figure 6.12 has been drawn to emphasize the algebraic decoding steps—calculation of syndrome, error pattern, and corrected output. In the real world, an (n, k) code is usually configured in systematic form. The decoder would not need to deliver the entire codeword; its output would consist of the data bits only. Hence, the Figure 6.12 circuitry becomes simplified by eliminating the gates that are shown with shading. For longer codes, such an implementation is very complex, and the preferred decoding techniques conserve circuitry by using a sequential approach instead of this parallel method [4]. It is also important to emphasize that Figure 6.12 has been configured to only detect and correct *single-error patterns* for the (6, 3) code. Error control for a double-error pattern would require additional circuitry.

6.4.9.1 Vector Notation

Codewords, error patterns, received vectors, and syndromes have been denoted by the vectors **U**, **e**, **r**, and **S** respectively. For notational simplicity, an index to denote a particular vector has generally been left off. However, to be precise, each of these vectors **U**, **e**, **r**, and **S** is one of a set having the general form

$$\mathbf{x}_j = \{x_1, x_2, \cdots, x_i, \cdots\}$$

Consider the range of the indices j and i in the context of the (6, 3) code in Table 6.1. For the codeword $\mathbf{U}_j$, the index $j = 1, \ldots, 2^k$ indicates that there are $2^3 = 8$ distinct codewords, and the index $i = 1, \ldots, n$ indicates that each codeword is made up of $n = 6$ bits. For a correctable error pattern $\mathbf{e}_j$, the index $j = 1, \ldots, 2^{n-k}$ indicates that there are $2^3 = 8$ coset leaders (7 nonzero correctable error patterns), and the index $i = 1, \ldots, n$ indicates that each error pattern is made up of $n = 6$ bits. For the received vector $\mathbf{r}_j$, the index $j = 1, \ldots, 2^n$ indicates that there are $2^6 = 64$ possible n-tuples that can be received, and the index $i = 1, \ldots, n$ indicates that each received n-tuple is made up of $n = 6$ bits. Finally, for the syndrome $\mathbf{S}_j$, the index $j = 1, \ldots, 2^{n-k}$ indicates that there are $2^3 = 8$ distinct syndrome vectors, and the index $i = 1, \ldots, n - k$ indicates that each syndrome is made up of $n - k = 3$ bits. In this chapter, the index is often dropped, and the vectors $\mathbf{U}_j$, $\mathbf{e}_j$, $\mathbf{r}_j$, and $\mathbf{S}_j$ are denoted as $\mathbf{U}$, $\mathbf{e}$, $\mathbf{r}$, and $\mathbf{S}$, respectively. The reader must be aware that for these vectors, an index is always inferred, even when it has been left off for notational simplicity.

6.5 ERROR-DETECTING AND CORRECTING CAPABILITY

6.5.1 Weight and Distance of Binary Vectors

It should be clear that not all error patterns can be correctly decoded. The error correction capability of a code will be investigated by first defining its structure. The *Hamming weight* $w(\mathbf{U})$ of a codeword $\mathbf{U}$ is defined to be the number of nonzero elements in $\mathbf{U}$. For a binary vector this is equivalent to the number of ones in the vector. For example, if $\mathbf{U} = 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1$, then $w(\mathbf{U}) = 5$. The *Hamming distance* between two codewords $\mathbf{U}$ and $\mathbf{V}$, denoted $d(\mathbf{U}, \mathbf{V})$, is defined to be the number of elements in which they differ—for example,

$$\begin{aligned} \mathbf{U} &= 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1 \\ \mathbf{V} &= 0\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \\ d(\mathbf{U}, \mathbf{V}) &= 6 \end{aligned}$$

By the properties of modulo-2 addition, we note that the sum of two binary vectors is another vector whose binary ones are located in those positions in which the two vectors differ—for example

$$\mathbf{U} + \mathbf{V} = 1\ 1\ 1\ 0\ 1\ 1\ 0\ 0\ 1$$

Thus, we observe that the Hamming distance between two codewords is equal to the Hamming weight of their sum: that is, $d(\mathbf{U}, \mathbf{V}) = w(\mathbf{U} + \mathbf{V})$. Also we see that the Hamming weight of a codeword is equal to its Hamming distance from the all-zeros vector.

6.5.2 Minimum Distance of a Linear Code

Consider the set of distances between all pairs of codewords in the space V_n. The smallest member of the set is the *minimum distance* of the code and is denoted $d_{\min}$. Why do you suppose we have an interest in the minimum distance; why not the maximum distance? The minimum distance, like the weakest link in a chain, gives us a measure of the code's minimum capability and therefore characterizes the code's strength.

As discussed earlier, the sum of any two codewords yields another codeword member of the subspace. This property of linear codes is stated simply as: If $\mathbf{U}$ and $\mathbf{V}$ are codewords, then $\mathbf{W} = \mathbf{U} + \mathbf{V}$ must also be a codeword. Hence the distance between two codewords is equal to the weight of a third codeword; that is, $d(\mathbf{U}, \mathbf{V}) = w(\mathbf{U} + \mathbf{V}) = w(\mathbf{W})$. Thus the minimum distance of a linear code can be ascertained without examining the distance between all combinations of codeword pairs. We only need to examine the weight of each codeword (excluding the all-zeros codeword) in the subspace; the minimum weight corresponds to the minimum distance, $d_{\min}$. Equivalently, $d_{\min}$ corresponds to the smallest of the set of distances between the all-zeros codeword and all the other codewords.

6.5.3 Error Detection and Correction

The task of the decoder, having received the vector $\mathbf{r}$, is to estimate the transmitted codeword $\mathbf{U}_i$. The optimal decoder strategy can be expressed in terms of the *maximum likelihood* algorithm (see Appendix B) as follows: Decide in favor of $\mathbf{U}_i$ if

$$P(\mathbf{r}|\mathbf{U}_i) = \max_{\text{over all } \mathbf{U}_j} P(\mathbf{r}|\mathbf{U}_j) \tag{6.41}$$

Since for the binary symmetric channel (BSC), the likelihood of $\mathbf{U}_i$ with respect to $\mathbf{r}$ is inversely proportional to the distance between $\mathbf{r}$ and $\mathbf{U}_i$, we can write: Decide in favor of $\mathbf{U}_i$ if

$$d(\mathbf{r}, \mathbf{U}_i) = \min_{\text{over all } \mathbf{U}_j} d(\mathbf{r}, \mathbf{U}_j) \tag{6.42}$$

In other words, the decoder determines the distance between $\mathbf{r}$ and each of the possible transmitted codewords $\mathbf{U}_j$, and selects as most likely a $\mathbf{U}_i$ for which

$$d(\mathbf{r}, \mathbf{U}_i) \leq d(\mathbf{r}, \mathbf{U}_j) \qquad \text{for } i, j = 1, \ldots, M \qquad \text{and} \qquad i \neq j \tag{6.43}$$

where $M = 2^k$ is the size of the codeword set. If the minimum is not unique, the choice between minimum distance codewords is arbitrary. Distance metrics are treated further in Chapter 7.

In Figure 6.13 the distance between two codewords $\mathbf{U}$ and $\mathbf{V}$ is shown using a number line calibrated in *Hamming distance.* Each black dot represents a corrupted codeword. Figure 6.13a illustrates the reception of vector $\mathbf{r}_1$, which is distance 1 from $\mathbf{U}$ and distance 4 from $\mathbf{V}$. An error-correcting decoder, following the

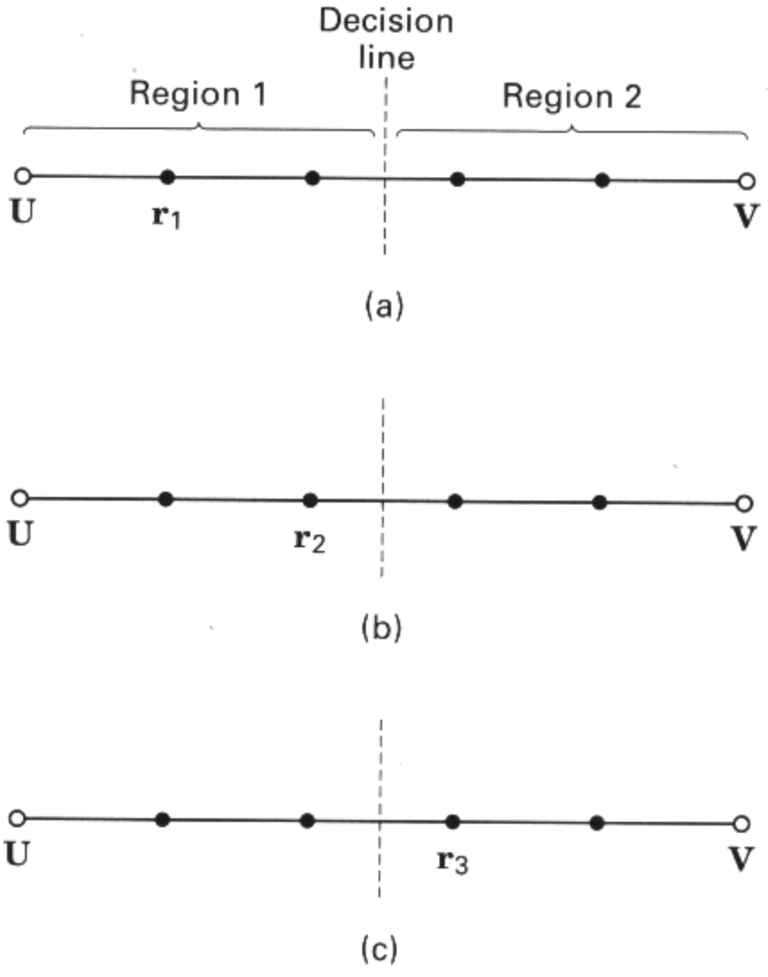


Figure 6.13 Error correction and detection capability. (a) Received vector $\mathbf{r}_1$. (b) Received vector $\mathbf{r}_2$. (c) Received vector $\mathbf{r}_3$.

maximum likelihood strategy, will select **U** upon receiving $\mathbf{r}_1$. If $\mathbf{r}_1$ had been the result of a 1-bit corruption to the transmitted code vector **U**, the decoder has successfully corrected the error. But if $\mathbf{r}_1$ had been the result of a 4-bit corruption to the transmitted code vector **V**, the result is a decoding error. Similarly, a double error in transmission of **U** might result in the received vector $\mathbf{r}_2$, which is distance 2 from **U** and distance 3 from **V**, as shown in Figure 6.13b. Here, too, the decoder will select **U** upon receiving $\mathbf{r}_2$. A triple error in transmission of **U** might result in a received vector $\mathbf{r}_3$ that is distance 3 from **U** and distance 2 from **V**, as shown in Figure 6.13c. Here the decoder will select **V** upon receiving $\mathbf{r}_3$, and will have made an error in decoding. From Figure 6.13 it should be clear that if error detection and not correction is the task, a corrupted vector—characterized by a black dot and representing a 1-bit, 2-bit, 3-bit, or 4-bit error—can be detected. However, five errors in transmission might result in codeword **V** being received when codeword **U** was actually transmitted; such as error would be *undetectable.*

From Figure 6.13 we can see that the error-detecting and error-correcting capabilities of a code are related to the *minimum distance* between codewords. The decision line in the figure serves the same purpose in the process of decoding as it does in demodulation, to define the decision regions. In the Figure 6.13 example, the decision criterion of choosing **U** if **r** falls in region 1, and choosing **V** if **r** falls in region 2, illustrates that such a code (with $d_{\min} = 5$) can correct two errors. In general, the *error-correcting capability* t of a code is defined as the maximum number of guaranteed correctable errors per codeword, and is written [4]

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \tag{6.44}$$

where $\lfloor x \rfloor$ means the largest integer not to exceed x. Often, a code that corrects all possible sequences of t or fewer errors can also correct certain sequences of $t + 1$ errors. This can be seen in Figure 6.11. In this example $d_{\min} = 3$, and thus from Equation (6.44), we can see that *all* $t = 1$ bit-error patterns are correctable. Also, *a single* $t + 1$ or 2-bit error pattern is correctable. In general, a t-error-correcting (n, k) linear code is capable of correcting a total of 2^{n-k} error patterns. If a t-error-correcting block code is used strictly for error correction on a binary symmetric channel (BSC) with transition probability p, the message-error probability, P_M, that the decoder commits an erroneous decoding, and that the n-bit block is in error, can be calculated by using Equation (6.18) as an upper bound:

$$P_M \leq \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j} \tag{6.45}$$

The bound becomes an equality when the decoder corrects all combinations of errors up to and including t errors, but no combinations of errors greater than t. Such decoders are called *bounded distance decoders.* The decoded bit-error probability, P_B, depends on the particular code and decoder. It can be expressed [5] by the following approximation:

$$P_B \approx \frac{1}{n} \sum_{j=t+1}^{n} j \binom{n}{j} p^j (1-p)^{n-j} \tag{6.46}$$

A block code needs to detect errors prior to correcting them. Or, it may be used for error-detection only. It should be clear from Figure 6.13 that any received vector characterized by a black dot (a corrupted codeword) can be identified as an error. Therefore, the error-detecting capability is defined in terms of $d_{\min}$ as

$$e = d_{\min} - 1 \tag{6.47}$$

A block code with minimum distance $d_{\min}$ guarantees that all error patterns of $d_{\min} - 1$ or fewer errors can be detected. Such a code is also capable of detecting a large fraction of error patterns with $d_{\min}$ or more errors. In fact, an (n, k) code is capable of detecting $2^n - 2^k$ error patterns of length n. The reasoning is as follows. There are a total of $2^n - 1$ possible nonzero error patterns in the space of 2^n n-tuples. Even the bit pattern of a valid codeword represents a potential error pattern. Thus there are $2^k - 1$ error patterns that are identical to the $2^k - 1$ nonzero codewords. If any of these $2^k - 1$ error patterns occurs, it alters the transmitted codeword $\mathbf{U}_i$ into another codeword $\mathbf{U}_j$. Thus $\mathbf{U}_j$ will be received and its syndrome is zero. The decoder accepts $\mathbf{U}_j$ as the transmitted codeword and thereby commits an incorrect decoding. Therefore, there are $2^k - 1$ undetectable error patterns. If the error pattern is not identical to one of the 2^k codewords, the syndrome test on the received vector $\mathbf{r}$ yields a nonzero syndrome, and the error is detected. Therefore, there are exactly $2^n - 2^k$ detectable error patterns. For large n, where $2^k << 2^n$, only a small fraction of error patterns are undetected.

6.5.3.1 Codeword Weight Distribution

Let A_j be the number of codewords of weight j within an (n, k) linear code. The numbers $A_0, A_1, \ldots, A_n$ are called the *weight distribution* of the code. If the code is used only for error detection, on a BSC, the probability that the decoder does not detect an error can be computed from the weight distribution of the code [5] as

$$P_{\text{nd}} = \sum_{j=1}^{n} A_j p^j (1 - p)^{n-j} \tag{6.48}$$

where p is the transition probability of the BSC. If the minimum distance of the code is $d_{\min}$, the values of A_1 to $A_{d_{\min}-1}$ are zero.

Example 6.5 Probability of an Undetected Error in an Error Detecting Code

Consider that the (6, 3) code, given in Section 6.4.3, is used only for error detection. Calculate the probability of an undetected error if the channel is a BSC and the transition probability is 10^{-2}.

Solution

The weight distribution of this code is $A_0 = 1, A_1 = A_2 = 0, A_3 = 4, A_4 = 3, A_5 = 0, A_6 = 0$. Therefore, we can write, using Equation (6.48),

$$P_{\text{nd}} = 4p^3(1 - p)^3 + 3p^4(1 - p)^2$$

For $p = 10^{-2}$, the probability of an undetected error is 3.9×10^{-6}.

6.5.3.2 Simultaneous Error Correction and Detection

It is possible to trade correction capability from the maximum guaranteed (t) where t is defined in Equation (6.44), for the ability to simultaneously detect a class of errors. A code can be used for the simultaneous correction of α errors and detection of β errors, where $\beta \geq \alpha$, provided that its minimum distance is [4]

$$d_{\min} \geq \alpha + \beta + 1 \tag{6.49}$$

When t or fewer errors occur, the code is capable of detecting and correcting them. When more than t but fewer than $e + 1$ errors occur, where e is defined in Equation (6.47), the code is capable of detecting their presence but correcting only a subset of them. For example, a code with $d_{\min} = 7$ can be used to simultaneously detect and correct in any one of the following ways:

Detect (β)	Correct (α)
3	3
4	2
5	1
6	0

Note that correction implies prior detection. For the above example, when there are three errors, all of them can be detected and corrected. When there are five errors, all of them can be detected but only a subset of them (one) can be corrected.

6.5.4 Visualization of a 6-Tuple Space

Figure 6.14 is a visualization of the eight codewords from the example of Section 6.4.3. The codewords are generated from linear combinations of the three independent 6-tuples in Equation (6.26); the codewords form a three-dimensional subspace. The figure shows such a subspace completely occupied by the eight codewords (large black circles); the coordinates of the subspace have purposely been drawn to emphasize their nonorthogonality. Figure 6.14 is an attempt to illustrate the entire space, containing sixty-four 6-tuples, even though there is no

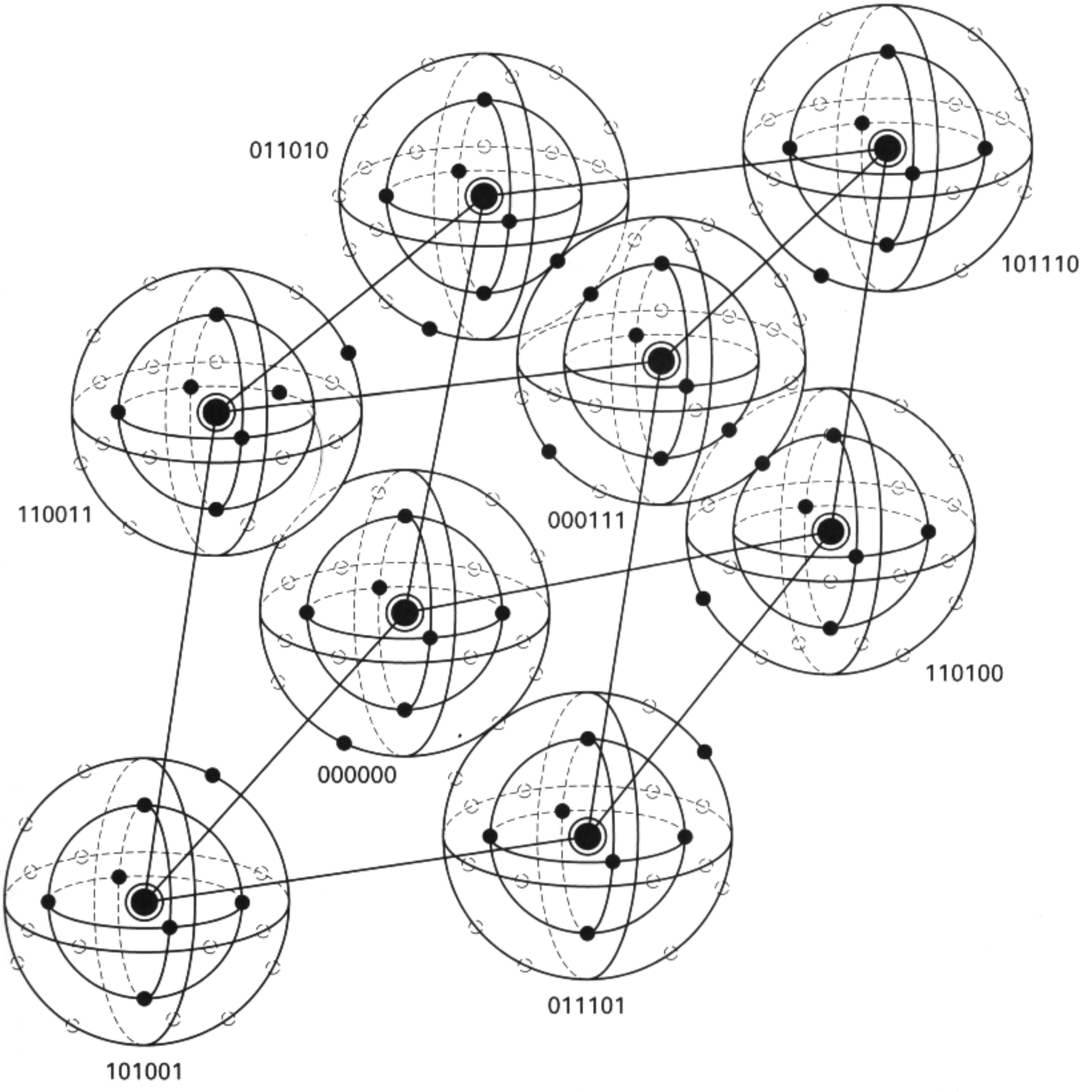


Figure 6.14 Example of eight codewords in a 6-tuple space.

precise way to draw or construct such a model. Spherical layers or shells are shown around each codeword. Each of the nonintersecting inner layers is a Hamming distance of 1 from its associated codeword; each outer layer is a Hamming distance of 2 from its codeword. Larger distances are not useful in this example. For each codeword, the two layers shown are occupied by perturbed codewords. There are six such points on each inner sphere (a total of 48 points), representing the six possible 1-bit error-perturbed vectors associated with each codeword. These 1-bit perturbed codewords are distinct in the sense that they can best be associated with only one codeword, and therefore can be corrected. As is seen from the standard array of Figure 6.11, there is also one 2-bit error pattern that can be corrected. There is a total of $\binom{6}{2} = 15$ different 2-bit error patterns that can be inflicted on each codeword, but only one of them, in our example the 0 1 0 0 0 1 error pattern, can be corrected. The other fourteen 2-bit error patterns yield vectors that cannot be uniquely identified with just one codeword; these noncorrectable error patterns yield vectors that are equivalent to the error-perturbed vectors of two or more codewords. In the figure, all correctable (fifty-six) 1- and 2-bit error-perturbed codewords are shown as small black circles. Perturbed codewords that cannot be corrected are shown as small clear circles.

Figure 6.14 is useful for visualizing the properties of a class of codes known as *perfect codes.* A t-error-correcting code is called a perfect code if its standard array has all the error patterns of t and fewer errors and no others as coset leaders (no residual error-correcting capacity). In terms of Figure 6.14, a t-error-correcting perfect code is one that can, with maximum likelihood decoding, correct all perturbed codewords occupying a shell at Hamming distance t or less from its originating codeword, and cannot correct any perturbed vectors occupying shells at distances greater than t.

Figure 6.14 is also useful for understanding the basic goal in the search for good codes. We would like for the space to be filled with as many codewords as possible (efficient utilization of the added redundancy), and we would also like these codewords to be as far away from one another as possible. Obviously, these goals conflict.

6.5.5 Erasure Correction

A receiver may be designed to declare a symbol *erased* when it is received ambiguously or when the receiver recognizes the presence of interference or a transient malfunction. Such a channel has an input alphabet of size Q and an output alphabet of size $Q + 1$; the extra output symbol is called an *erasure flag,* or simply an *erasure.* When a demodulator makes a symbol error, two parameters are needed to correct that error, its *location* and its *correct* symbol value. In the case of binary symbols, this reduces to needing only the error location. However, if the demodulator declares a symbol *erased,* although the correct symbol value is not known, the symbol location *is* known, and for this reason, the decoding of erased codewords can be simpler than error correcting. An error control code can be used to correct erasures or to correct errors and erasures simultaneously. If the code has minimum distance $d_{\min}$, any pattern of ρ or fewer erasures can be corrected if [6]

$$d_{\min} \geq \rho + 1 \tag{6.50}$$

Assume for the moment that no errors occur outside the erasure positions. The advantage of correcting by means of erasures is expressed quantitatively as follows: If a code has a minimum distance $d_{\min}$, then from Equation (6.50), $d_{\min} - 1$ erasures can be reconstituted. Since the number of errors that can be corrected without erasure information is $(d_{\min} - 1)/2$ at most, from Equation (6.44), the advantage of correcting by means of erasures is clear. Further, any pattern of α errors and γ erasures can be corrected simultaneously if [6]

$$d_{\min} \geq 2\alpha + \gamma + 1 \tag{6.51}$$

Simultaneous erasure correction and error correction can be accomplished in the following way. First, the γ-erased positions are replaced with zeros and the resulting codeword is decoded normally. Next, the γ-erased positions are replaced with ones, and the decoding operation is repeated on this version of the codeword. Of the two codewords obtained (one with erasures replaced by zeros, and the other with erasures replaced by ones) the one corresponding to the smallest number of errors corrected outside the γ-erased positions is selected. This technique will always result in correct decoding if Equation (6.51) is satisfied.

Example 6.6 Erasure Correction

Consider the codeword set presented in Section 6.4.3:

000000 110100 011010 101110 101001 011101 110011 000111

Suppose that the codeword 110011 was transmitted and that the two leftmost digits were declared by the receiver to be erasures. Verify that the received flawed sequence xx0011 can be corrected.

Solution

Since $d_{\min} = \rho + 1 = 3$, the code can correct as many as $\rho = 2$ erasures. This is easily verified above or with Figure 6.11 by comparing the rightmost four digits of xx0011 with each of the allowable codewords. The codeword that was actually transmitted is closest in Hamming distance to the flawed sequence.

6.6 USEFULNESS OF THE STANDARD ARRAY

6.6.1 Estimating Code Capability

The standard array can be thought of as an organizational tool, a filing cabinet that contains all of the possible 2^n entries in the space of n-tuples—nothing missing, and nothing replicated. At first glance, the benefits of this tool seem limited to *small* block codes, because for code lengths beyond $n = 20$, there are millions of n-tuples in the space. However, even for large codes, the standard array allows visualization of important performance issues, such as possible trade-offs between error correction and detection, as well as bounds on error-correction capability. One such bound, called the *Hamming bound* [7], is described as

$$\text{Number of parity bits:} \quad n - k \geq \log_2\left[1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}\right] \tag{6.52a}$$

or

$$\text{Number of cosets:} \quad 2^{n-k} \geq \left[1 + \binom{n}{1} + \binom{n}{2} + \cdots + \binom{n}{t}\right] \tag{6.52b}$$

where $\binom{n}{j}$, defined in Equation (6.16), represents the number of ways in which j bits out of n may be in error. Note that the sum of the terms within the square brackets of Equation (6.52) yields the minimum number of rows needed in the standard array to correct all combinations of errors through t-bit errors. The inequality gives a lower bound on $n - k$, the number of parity bits (or the number of 2^{n-k} cosets) as a function of the t-bit error-correction capability of the code. Similarly, the inequality can be described as giving an upper bound on the t-bit error correction capability as a function of the number of $n - k$ parity bits (or 2^{n-k} cosets). For any (n, k) linear block code to provide a t-bit error-correcting capability, it is a necessary condition that the Hamming bound be met.

To demonstrate how the standard array provides a visualization of this bound, let us choose the (127, 106) BCH code as an example. The array contains all $2^n = 2^{127} \approx 1.70 \times 10^{38}$ n-tuples in the space. The topmost row of the array contains the $2^k = 2^{106} \approx 8.11 \times 10^{31}$ codewords; hence, this is the number of columns in the array. The leftmost column contains the $2^{n-k} = 2^{21} = 2{,}097{,}152$ coset leaders; hence, this is the number of rows in the array. Although the number of n-tuples and codewords is enormous, the concern is not with any individual entry; the primary interest is in the number of cosets. There are 2,097,152 cosets, and hence there are at most 2,097,151 error patterns that can be corrected by this code. Next, it is shown how this number of cosets dictates an upper bound on the t-bit error correcting capability of the code.

Since each codeword contains 127 bits, there are 127 ways to make single errors. We next compute how many ways there are to make double errors, namely $\binom{127}{2} = 8{,}001$. We move on to triple errors because thus far only a small portion of the total 2,097,151 correctable error-patterns have been used. There are $\binom{127}{3} = 333{,}375$ ways to make triple errors. On Table 6.3 these computations are listed, indicating that the all-zeros error pattern requires the presence of the first coset, followed by single, double, and triple errors. Also shown are the number of cosets required for each error type and the cumulative number of cosets necessary through that error type. From this table it can be seen that a (127, 106) code can correct all single-, double-, and triple-error patterns, which only account for 341,504 of the available 2,097,152 cosets. The unused 1,755,648 rows are indicative of the fact that more error correction is possible. Indeed, it might be tempting to try fitting all possible 4-bit error patterns into the array. However, from Table 6.3 it can be seen that this is not possible, because the number of remaining cosets in the array is much smaller than the cumulative number of cosets required for correcting 4-bit errors, as indicated by the last line of the table. Therefore, for this (127, 106) example, the code has a Hamming bound that guarantees the correction of up to and including all 3-bit errors.

TABLE 6.3 Error-Correction Bound for the (127, 106) Code

Number of Bit Errors	Number of Cosets Required	Cumulative Number of Cosets Required
0	1	1
1	127	128
2	8,001	8,129
3	333,375	341,504
4	10,334,625	10,676,129

6.6.2 An (*n*, *k*) Example

The standard array provides insight into the trade-offs that are possible between error correction and detection. Consider a new (n, k) code example, and the factors that dictate what values of (n, k) should be chosen.

1. In order to perform a nontrivial trade-off between error correction and error detection, it is desired that the code have an error-correcting capability of at least $t = 2$. We use Equation (6.44) for finding the minimum distance, as follows: $d_{\min} = 2t + 1 = 5$.
2. For a nontrivial code system, it is desired that the number of data bits be at least $k = 2$. Thus, there will be $2^k = 4$ codewords. The code can now be designated as an $(n, 2)$ code.
3. We look for the minimum value of n that will allow correcting all possible single and double errors. In this example, each of the 2^n n-tuples in the array will be tabulated. The minimum value of n is desired because whenever n is incremented by just a single integer, the number of n-tuples in the standard array doubles. It is, of course, desired that the list be of manageable size. For real world codes, we want the minimum n for different reasons—bandwidth efficiency and simplicity. If the Hamming bound is used in choosing n, then $n = 7$ could be selected. However, the dimensions of such a (7, 2) code will not meet our stated requirements of t = 2-bit error-correction capability and $d_{\min} = 5$. To see this, it is necessary to introduce another upper bound on the t-bit error correction capability (or $d_{\min}$). This bound, called the *Plotkin bound* [7], is described by

$$d_{\min} \leq \frac{n \times 2^{k-1}}{2^k - 1} \tag{6.53}$$

In general, a linear (n, k) code must meet all upper bounds involving error-correction capability (or minimum distance). For high-rate codes, if the Hamming bound is met, then the Plotkin bound will also be met; this was the case for the earlier (127, 106) code example. For low rate codes, it is the other way around [7]. Since this example entails a low-rate code, it is important to test error-correction capability via the Plotkin bound. Because $d_{\min} = 5$, it should be clear from Equation (6.53) that n must be 8, and therefore, the minimum dimensions of the code are (8, 2), in order to meet the requirements for this example. Would anyone use such

a low-rate double-error correcting code as this (8, 2) code? No, it would be too bandwidth expansive compared with more efficient codes that are available. It is used here for tutorial purposes, because its standard array is of manageable size.

6.6.3 Designing the (8, 2) Code

A natural question to ask is, How does one select codewords out of the space of 2^8 8-tuples? There is not a single solution, but there are constraints in how choices are made. The following are the elements that help point to a solution:

1. The number of codewords is $2^k = 2^2 = 4$.
2. The all-zeros vector must be one of the codewords.
3. The property of closure must apply. This property dictates that the sum of any two codewords in the space must yield a valid codeword in the space.
4. Each codeword is 8 bits long.
5. Since $d_{\min} = 5$, the weight of each codeword (except for the all-zeros codeword) must also be at least 5 (by virtue of the closure property). The weight of a vector is defined as the number of nonzero components in the vector.
6. Assume that the code is systematic, and thus the rightmost 2 bits of each codeword are the corresponding message bits.

The following is a candidate assignment of codewords to messages that meets all of the preceeding conditions:

Messages	Codewords
00	00000000
01	11110001
10	00111110
11	11001111

The design of the codeword set can begin in a very arbitrary way; it is only necessary to adhere to the properties of weight and systematic form of the code. The selection of the first few codewords is often simple. However, as the process continues, the selection routine becomes harder, and the choices become more constrained because of the need to adhere to the closure property.

6.6.4 Error Detection versus Error Correction Trade-offs

For the (8, 2) code system selected in the previous section, the $(k \times n) = (2 \times 8)$ generator matrix can be written as

$$\mathbf{G} = \begin{bmatrix} 0\,0\,1\,1\,1\,1\,1\,0 \\ 1\,1\,1\,1\,0\,0\,0\,1 \end{bmatrix}$$

Decoding starts with the computation of a syndrome, which can be thought of as learning the "symptom" of an error. For an (n, k) code, an $(n - k)$-bit syndrome $\mathbf{S}$ is

the product of an n-bit received vector $\mathbf{r}$, and the transpose of an $(n-k) \times n$ parity-check matrix $\mathbf{H}$, where $\mathbf{H}$ is constructed so that the rows of $\mathbf{G}$ are orthogonal to the rows of $\mathbf{H}$; that is $\mathbf{G}\mathbf{H}^T = \mathbf{0}$. For this (8, 2) example, $\mathbf{S}$ is a 6-bit vector, and $\mathbf{H}$ is a 6×8 matrix, where

$$\mathbf{H}^t = \begin{bmatrix} 1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,1 \\ 0\,0\,1\,1\,1\,1 \\ 1\,1\,1\,1\,0\,0 \end{bmatrix}$$

The syndrome for each error pattern can be calculated using Equation (6.37), namely

$$\mathbf{S}_i = \mathbf{e}_i \mathbf{H}^T \qquad i = 1, \cdots, 2^{n-k}$$

where $\mathbf{S}_i$ is one of the $2^{n-k} = 64$ syndromes, and $\mathbf{e}_i$ is one of the 64 coset leaders (error patterns) in the standard array. Figure 6.15 shows a tabulation of the standard array as well as all 64 syndromes for the (8, 2) code. The set of syndromes were calculated by using Equation (6.37); the entries of any given row (coset) of the standard array have the same syndrome. The correction of a corrupted codeword proceeds by computing its syndrome and locating the error pattern that corresponds to that syndrome. Finally, the error pattern is modulo-2 added to the corrupted codeword yielding the corrected output. Equation (6.49), repeated below, indicates that error-detection and error-correction capabilities can be traded, provided that the distance relationship

$$d_{\min} \geq \alpha + \beta + 1$$

prevails. Here, α represents the number of bit errors to be corrected, β represents the number of bit errors to be detected, and $\beta \geq \alpha$. The trade-off choices available for the (8, 2) code example are as follows:

Detect (β)	Correct (α)
2	2
3	1
4	0

This table shows that the (8, 2) code can be implemented to perform only error correction, which means that it first detects as many as $\beta = 2$ errors and then corrects them. If some error correction is sacrificed so that the code will only correct single errors, then the detection capability is increased so that all $\beta = 3$ errors can be detected. Finally, if error correction is completely sacrificed, the decoder can be implemented so that all $\beta = 4$ errors can be detected. In the case of error detection

Syndromes		Standard array			
000000	1.	00000000	11110001	00111110	11001111
111100	2.	00000001	11110000	00111111	11001110
001111	3.	00000010	11110011	00111100	11001101
000001	4.	00000100	11110101	00111010	11001011
000010	5.	00001000	11111001	00110110	11000111
000100	6.	00010000	11100001	00101110	11011111
001000	7.	00100000	11010001	00011110	11101111
010000	8.	01000000	10110001	01111110	10001111
100000	9.	10000000	01110001	10111110	01001111
110011	10.	00000011	11110010	00111101	11001100
111101	11.	00000101	11110100	00111011	11001010
111110	12.	00001001	11111000	00110111	11000110
111000	13.	00010001	11100000	00101111	11011110
110100	14.	00100001	11010000	00011111	11101110
101100	15.	01000001	10110000	01111111	10001110
011100	16.	10000001	01110000	10111111	01001110
001110	17.	00000110	11110111	00111000	11001001
001101	18.	00001010	11111011	00110100	11000101
001011	19.	00010010	11100011	00101100	11011101
000111	20.	00100010	11010011	00011100	11101101
011111	21.	01000010	10110011	01111100	10001101
101111	22.	10000010	01110011	10111100	01001101
000011	23.	00001100	11111101	00110010	11000011
000101	24.	00010100	11100101	00101010	11011011
001001	25.	00100100	11010101	00011010	11101011
010001	26.	01000100	10110101	01111010	10001011
100001	27.	10000100	01110101	10111010	01001011
000110	28.	00011000	11101111	00100110	11010111
001010	29.	00101000	11011001	00010110	11100111
010010	30.	01001000	10111001	01110110	10000111
100010	31.	10001000	01111001	10110110	01000111
001100	32.	00110000	11000001	00001110	11111111
010100	33.	01010000	10100001	01101110	10011111
100100	34.	10010000	01100001	10101110	01011111
011000	35.	01100000	10010001	01011110	10101111
101000	36.	10100000	01010001	10011110	01101111
110000	37.	11000000	00110001	11111110	00001111
110010	38.	00000111	11110110	00111001	11001000
110111	39.	00010011	11100010	00101101	11011100
111011	40.	00100011	11010010	00011101	11101100
100011	41.	01000011	10110010	01111101	10001100
010011	42.	10000011	01110010	10111101	01001100
111111	43.	00001101	11111100	00110011	11000010
111001	44.	00010101	11100100	00101011	11011010
110101	45.	00100101	11010100	00011011	11101010
101101	46.	01000101	10110100	01111011	10001010
011101	47.	10000101	01110100	10111011	01001010
011110	48.	01000110	10110111	01111000	10001001
101110	49.	10000110	01110111	10111000	01001001
100101	50.	10010100	01100101	10101010	01011011
011001	51.	01100100	10010101	01011010	10101011
110001	52.	11000100	00110101	11111010	00001011
011010	53.	01101000	10011001	01010110	10100111
010110	54.	01011000	10101001	01100110	10010111
100110	55.	10011000	01101001	10100110	01010111
101010	56.	10101000	01011001	10010110	01100111
101001	57.	10100100	01010101	10011010	01101011
100111	58.	10100010	01010011	10011100	01101101

Figure 6.15 The syndromes and the standard array for the (8, 2) code.

only, the circuitry is very simple. The syndrome is computed and an error is detected whenever a nonzero syndrome occurs.

For correcting single errors, the decoder can be implemented with gates [4], similar to the circuitry in Figure 6.12, where a received code vector $\mathbf{r}$ enters at two places. In the top part of the figure, the received digits are connected to exclusive-OR gates, which yield the syndrome. For any given received vector, the syndrome is obtained from Equation (6.35) as

$$\mathbf{S}_i = \mathbf{r}_i \mathbf{H}^T \qquad i = 1, \cdots, 2^{n-k}$$

Using the $\mathbf{H}^T$ values for the (8, 2) code, the wiring between the received digits and the exclusive-OR gates in a circuit similar to the one in Figure 6.12, must be connected to yield

$$\mathbf{S}_i = [r_1 \ r_2 \ r_3 \ r_4 \ r_5 \ r_6 \ r_7 \ r_8] \begin{bmatrix} 1\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,1 \\ 0\,0\,1\,1\,1\,1 \\ 1\,1\,1\,1\,0\,0 \end{bmatrix}$$

Each of the s_j digits ($j = 1, \ldots, 6$) making up syndrome $\mathbf{S}_i$ ($i = 1, \ldots, 64$) is related to the input-received code vector in the following way:

$$s_1 = r_1 + r_8 \qquad s_2 = r_2 + r_8 \qquad s_3 = r_3 + r_7 + r_8$$
$$s_4 = r_4 + r_7 + r_8 \qquad s_5 = r_5 + r_7 \qquad s_6 = r_6 + r_7$$

To implement a decoder circuit similar to Figure 6.12 for the (8, 2) code necessitates that the eight received digits be connected to six modulo-2 adders yielding the syndrome digits as described above. Additional modifications to the figure need to be made accordingly.

If the decoder is implemented to correct only single errors; that is $\alpha = 1$ and $\beta = 3$, then this is tantamount to drawing a line under coset 9 in Figure 6.15, and error correction takes place only when one of the eight syndromes associated with a single error appears. The decoding circuitry (similar to Figure 6.12) then transforms the syndrome to its corresponding error pattern. The error pattern is then modulo-2 added to the "potentially" corrupted received vector, yielding the corrected output. Additional gates are needed to test for the case in which the syndrome is nonzero and there is no correction designed to take place. For single-error correction, such an event happens for any of the syndromes numbered 10 through 64. This outcome is then used to indicate an error detection.

If the decoder is implemented to correct single and double errors, which means that $\beta = 2$ errors are detected and then corrected, then this is tantamount to drawing a line under coset 37 in the standard array of Figure 6.15. Even though this (8, 2) code is capable of correcting some combination of triple errors corresponding

to the coset leaders 38 through 64, a decoder is most often implemented as a *bounded distance* decoder, which means that it corrects all combinations of errors up to and including t errors, but no combinations of errors greater than t. The unused error-correction capability can be applied toward some error-detection enhancement. As before, the decoder can be implemented with gates similar to those shown in Figure 6.12.

6.6.5 The Standard Array Provides Insight

In the context of Figure 6.15, the (8, 2) code satisfies the Hamming bound. That is, from the standard array it is recognizable that the (8, 2) code can correct all combinations of single and double errors. Consider the following question: Given that transmission takes place over a channel that always introduces errors in the form of a burst of 3-bit errors and thus there is no interest in correcting single or double errors, wouldn't it be possible to set up the coset leaders to correspond to only triple errors? It is simple to see that in a sequence of 8 bits there are $\binom{8}{3} = 56$ ways to make triple errors. If we only want to correct all these 56 combinations of triple errors, there is sufficient room (sufficient number of cosets) in the standard array, since there are 64 rows. Will that work? No, it will not. For any code, the overriding parameter for determining error-correcting capability is $d_{\min}$. For the (8, 2) code, $d_{\min} = 5$ dictates that only 2-bit error correction is possible.

How can the standard array provide some insight as to why this scheme won't work? In order for a group of x-bit error patterns to enable x-bit error correction, the entire group of weight-x vectors must be coset leaders; that is, they must only occupy the leftmost column. In figure 6.15, it can be seen that all weight-1 and weight-2 vectors appear in the leftmost column of the standard array, and nowhere else. Even if we forced all weight-3 vectors into row numbers 2 through 57, we would find that some of these vectors would have to reappear elsewhere in the array (which violates a basic property of the standard array). In Figure 6.15 a shaded box is drawn around every one of the 56 vectors having a weight of 3. Look at the coset leaders representing 3-bit error patterns, in rows 38, 41–43, 46–49, and 52 of the standard array. Now look at the entries of the same row numbers in the rightmost column, where shaded boxes indicate other weight-3 vectors. Do you see the ambiguity that exists for each of the rows listed above, and why it is not possible to correct all 3-bit error patterns with this (8, 2) code? Suppose the decoder receives the weight-3 vector 1 1 0 0 1 0 0 0, located at row 38 in the rightmost column. This flawed codeword could have arisen in one of two ways: One would be that codeword 1 1 0 0 1 1 1 1 was sent and the 3-bit error pattern 0 0 0 0 0 1 1 1 perturbed it; the other would be that codeword 0 0 0 0 0 0 0 0 was sent and the 3-bit error pattern 1 1 0 0 1 0 0 0 perturbed it.

6.7 CYCLIC CODES

Binary cyclic codes are an important subclass of linear block codes. The codes are easily implemented with feedback shift registers; the syndrome calculation is easily

accomplished with similar feedback shift registers; and the underlying algebraic structure of a cyclic code lends itself to efficient decoding methods. An (n, k) linear code is called a *cyclic code* if it can be described by the following property. If the n-tuple $\mathbf{U} = (u_0, u_1, u_2, \ldots, u_{n-1})$ is a codeword in the subspace S, then $\mathbf{U}^{(1)} = (u_{n-1}, u_0, u_1, u_2, \ldots, u_{n-2})$ obtained by an end-around shift, is also a codeword in S. Or, in general, $\mathbf{U}^{(i)} = (u_{n-i}, u_{n-i+1}, \ldots u_{n-1}, u_0, u_1, \ldots, u_{n-i-1})$, obtained by i end-around or cyclic shifts, is also a codeword in S.

The components of a codeword $\mathbf{U} = (u_0, u_1, u_2, \ldots, u_{n-1})$ can be treated as the coefficients of a polynomial $\mathbf{U}(X)$ as follows:

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-1}X^{n-1} \tag{6.54}$$

The polynomial function $\mathbf{U}(X)$ can be thought of as a "placeholder" for the digits of the codeword $\mathbf{U}$; that is, an n-tuple vector is described by a polynomial of degree $n - 1$ or less. The presence or absence of each term in the polynomial indicates the presence of a 1 or 0 in the corresponding location of the n-tuple. If the u_{n-1} component is nonzero, the polynomial is of degree $n - 1$. The usefulness of this polynomial description of a codeword will become clear as we discuss the algebraic structure of the cyclic codes.

6.7.1 Algebraic Structure of Cyclic Codes

Expressing the codewords in polynomial form, the cyclic nature of the code manifests itself in the following way. If $\mathbf{U}(X)$ is an $(n - 1)$-degree codeword polynomial, then $\mathbf{U}^{(i)}(X)$, the remainder resulting from dividing $X^i\mathbf{U}(X)$ by $X^n + 1$, is also a codeword; that is,

$$\frac{X^i\mathbf{U}(X)}{X^n + 1} = \mathbf{q}(X) + \frac{\mathbf{U}^{(i)}(X)}{X^n + 1} \tag{6.55a}$$

or, multiplying through by $X^n + 1$,

$$X^i\mathbf{U}(X) = \mathbf{q}(X)(X^n + 1) + \underbrace{\mathbf{U}^{(i)}(X)}_{\text{remainder}} \tag{6.55b}$$

which can also be described in terms of modulo arithmetic as

$$\mathbf{U}^{(i)}(X) = X^i\mathbf{U}(X) \text{ modulo } (X^n + 1) \tag{6.56}$$

where x modulo y is defined as the remainder obtained from dividing x by y. Let us demonstrate the validity of Equation (6.56) for the case of $i = 1$:

$$\mathbf{U}(X) = u_0 + u_1X + u_2X^2 + \cdots + u_{n-2}X^{n-2} + u_{n-1}X^{n-1}$$
$$X\mathbf{U}(X) = u_0X + u_1X^2 + u_2X^3 + \cdots + u_{n-2}X^{n-1} + u_{n-1}X^n$$

We now add and subtract u_{n-1}; or, since we are using modulo-2 arithmetic, we add u_{n-1} twice, as follows:

$$X\mathbf{U}(X) = \underbrace{u_{n-1} + u_0X + u_1X^2 + u_2X^3 + \cdots + u_{n-2}X^{n-1}}_{\mathbf{U}^{(1)}(X)} + u_{n-1}X^n + u_{n-1}$$

$$= \mathbf{U}^{(1)}(X) + u_{n-1}(X^n + 1)$$

Since $U^{(1)}(X)$ is of degree $n - 1$, it cannot be divided by $X^n + 1$. Thus, from Equation (6.55a), we can write

$$\mathbf{U}^{(1)}(X) = X\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

By extension, we arrive at Equation (6.56):

$$\mathbf{U}^{(i)}(X) = X^i\mathbf{U}(X) \text{ modulo } (X^n + 1)$$

Example 6.7 Cyclic Shift of a Code Vector

Let $\mathbf{U} = 1\ 1\ 0\ 1$, for $n = 4$. Express the codeword in polynomial form, and using Equation (6.56), solve for the third end-around shift of the codeword.

Solution

$$\mathbf{U}(X) = 1 + X + X^3 \qquad \text{(polynomial is written low order to high order);}$$
$$X^i\mathbf{U}(X) = X^3 + X^4 + X^6, \qquad \text{where } i = 3.$$

Divide $X^3\mathbf{U}(X)$ by $X^4 + 1$, and solve for the remainder using polynomial division:

$$\begin{array}{r} X^2 + 1 \\ X^4 + 1 \;\overline{)\,X^6 + X^4 + X^3} \\ \underline{X^6 \qquad\qquad + X^2} \\ X^4 + X^3 + X^2 \\ \underline{X^4 \qquad\qquad + 1} \\ X^3 + X^2 + 1 \end{array} \qquad \text{remainder } \mathbf{U}^{(3)}(X)$$

Writing the remainder low order to high order: $1 + X^2 + X^3$, the codeword $\mathbf{U}^{(3)} = 1\ 0\ 1\ 1$ is three cyclic shifts of $\mathbf{U} = 1\ 1\ 0\ 1$. Remember that for binary codes, the addition operation is performed modulo-2, so that $+ 1 = -1$, and we consequently do not show any minus signs in the computation.

6.7.2 Binary Cyclic Code Properties

We can generate a cyclic code using a *generator polynomial* in much the way that we generated a block code using a generator matrix. The generator polynomial $\mathbf{g}(X)$ for an (n, k) cyclic code is unique and is of the form

$$\mathbf{g}(X) = g_0 + g_1X + g_2X^2 + \cdots + g_pX^p \tag{6.57}$$

where g_0 and g_p must equal 1. Every codeword polynomial in the subspace is of the form $\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X)$, where $\mathbf{U}(X)$ is a polynomial of degree $n - 1$ or less. Therefore, the message polynomial $\mathbf{m}(X)$ is written as

$$\mathbf{m}(X) = m_0 + m_1X + m_2X^2 + \cdots + m_{n-p-1}X^{n-p-1} \tag{6.58}$$

There are 2^{n-p} codeword polynomials, and there are 2^k code vectors in an (n, k) code. Since there must be one codeword polynomial for each code vector,

$$n - p = k$$

or

$$p = n - k$$

Hence, $\mathbf{g}(X)$, as shown in Equation (6.57), must be of degree $n - k$, and every codeword polynomial in the (n, k) cyclic code can be expressed as

$$\mathbf{U}(X) = (m_0 + m_1X + m_2X^2 + \cdots + m_{k-1}X^{k-1})\mathbf{g}(X) \tag{6.59}$$

$\mathbf{U}$ is said to be a valid codeword of the subspace S *if, and only if,* $\mathbf{g}(X)$ divides into $\mathbf{U}(X)$ without a remainder.

A generator polynomial $\mathbf{g}(X)$ of an (n, k) cyclic code is a factor of $X^n + 1$; that is, $X^n + 1 = \mathbf{g}(X)\mathbf{h}(X)$. For example,

$$X^7 + 1 = (1 + X + X^3)(1 + X + X^2 + X^4)$$

Using $\mathbf{g}(X) = 1 + X + X^3$ as a generator polynomial of degree $n - k = 3$, we can generate an $(n, k) = (7, 4)$ cyclic code. Or, using $\mathbf{g}(X) = 1 + X + X^2 + X^4$ where $n - k = 4$ we can generate a $(7, 3)$ cyclic code. In summary, if $\mathbf{g}(X)$ is a polynomial of degree $n - k$ and is a factor of $X^n + 1$, then $\mathbf{g}(X)$ uniquely generates an (n, k) cyclic code.

6.7.3 Encoding in Systematic Form

In Section 6.4.5 we introduced the *systematic* form and discussed the reduction in complexity that makes this encoding form attractive. Let us use some of the algebraic properties of the cyclic code to establish a systematic encoding procedure. We can express the message vector in polynomial form, as follows:

$$\mathbf{m}(X) = m_0 = m_1X + m_2X^2 + \cdots + m_{k-1}X^{k-1} \tag{6.60}$$

In systematic form, the message digits are utilized as part of the codeword. We can think of shifting the message digits into the rightmost k stages of a codeword register, and then appending the parity digits by placing them in the leftmost $n - k$ stages. Therefore, we manipulate the message polynomial algebraically so that it is right-shifted $n - k$ positions. If we multiply $\mathbf{m}(X)$ by X^{n-k}, we get the right-shifted message polynomial:

$$X^{n-k}\mathbf{m}(X) = m_0X^{n-k} + m_1X^{n-k+1} + \cdots + m_{k-1}X^{n-1} \tag{6.61}$$

If we next divide Equation (6.61) by $\mathbf{g}(X)$, the result can be expressed as

$$X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{p}(X) \tag{6.62}$$

where the remainder $\mathbf{p}(X)$ can be expressed as

$$\mathbf{p}(X) = p_0 + p_1X + p_2X^2 + \cdots + p_{n-k-1}X^{n-k-1}$$

We can also say that

$$\mathbf{p}(X) = X^{n-k}\mathbf{m}(X) \text{ modulo } \mathbf{g}(X) \tag{6.63}$$

Adding $\mathbf{p}(X)$ to both sides of Equation (6.62), using modulo-2 arithmetic, we get

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X) = \mathbf{U}(X) \tag{6.64}$$

The left-hand side of Equation (6.64) is recognized as a valid codeword polynomial, since it is a polynomial of degree $n-1$ or less, and when divided by $\mathbf{g}(X)$ there is a zero remainder. This codeword can be expanded into its polynomial terms as follows:

$$\mathbf{p}(X) + X^{n-k}\mathbf{m}(X) = p_0 + p_1X + \cdots + p_{n-k-1}X^{n-k-1} + m_0X^{n-k} + m_1X^{n-k+1} + \cdots + m_{k-1}X^{n-1}$$

The codeword polynomial corresponds to the code vector

$$\mathbf{U} = \underbrace{(p_0, p_1, \ldots, p_{n-k-1}}_{(n-k)\text{ parity bits}}, \underbrace{m_0, m_1, \ldots, m_{k-1})}_{k\text{ message bits}} \tag{6.65}$$

Example 6.8 Cyclic Code in Systematic Form

Using the generator polynomial $\mathbf{g}(X) = 1 + X + X^3$, generate a systematic codeword from the (7, 4) codeword set for the message vector $\mathbf{m} = 1\ 0\ 1\ 1$.

Solution

$$\mathbf{m}(X) = 1 + X^2 + X^3, \quad n = 7, \quad k = 4, \quad n - k = 3;$$
$$X^{n-k}\mathbf{m}(X) = X^3(1 + X^2 + X^3) = X^3 + X^5 + X^6$$

Dividing $X^{n-k}\mathbf{m}(X)$ by $\mathbf{g}(X)$ using polynomial division, we can write

$$X^3 + X^5 + X^6 = \underbrace{(1 + X + X^2 + X^3)}_{\text{quotient } \mathbf{q}(X)} \underbrace{(1 + X + X^3)}_{\text{generator } \mathbf{g}(X)} + \underbrace{1}_{\text{remainder } \mathbf{p}(X)}$$

Using Equation (6.64) yields

$$\mathbf{U}(X) = \mathbf{p}(X) + X^3\mathbf{m}(X) = 1 + X^3 + X^5 + X^6$$

$$\mathbf{U} = \underbrace{1\ 0\ 0}_{\text{parity bits}} \quad \underbrace{1\ 0\ 1\ 1}_{\text{message bits}}$$

6.7.4 Circuit for Dividing Polynomials

We have seen that the cyclic shift of a codeword polynomial and that the encoding of a message polynomial involves the division of one polynomial by another. Such an operation is readily accomplished by a *dividing circuit* (feedback shift register). Given two polynomials $\mathbf{V}(X)$ and $\mathbf{g}(X)$, where

$$\mathbf{V}(X) = v_0 + v_1X + v_2X^2 + \cdots + v_mX^m$$

and

$$\mathbf{g}(X) = g_0 + g_1X + g_2X^2 + \cdots + g_pX^p$$

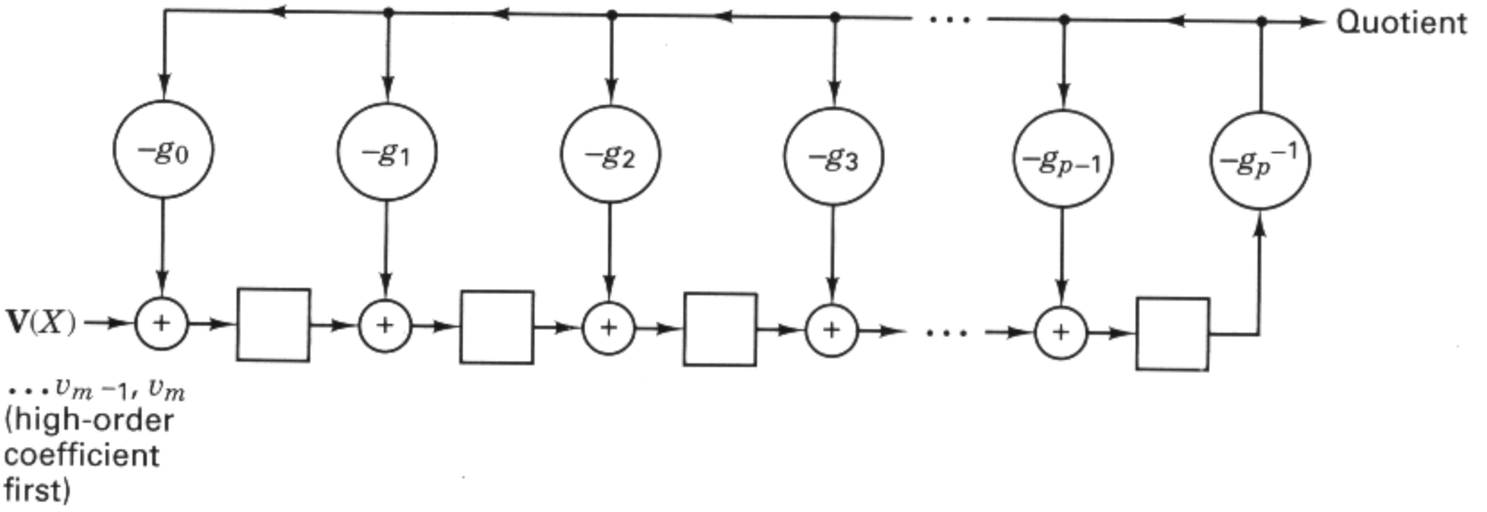


Figure 6.16 Circuit for dividing polynomials.

such that $m \geq p$, the divider circuit of Figure 6.16 performs the polynomial division steps of dividing $\mathbf{V}(X)$ by $\mathbf{g}(X)$, thereby determining the quotient and remainder terms:

$$\frac{\mathbf{V}(X)}{\mathbf{g}(X)} = \mathbf{q}(X) + \frac{\mathbf{p}(X)}{\mathbf{g}(X)}$$

The stages of the register are first initialized by being filled with zeros. The first p shifts enter the most significant (higher-order) coefficients of $\mathbf{V}(X)$. After the pth shift, the quotient output is $g_p^{-1} v_m$; this is the highest-order term in the quotient. For each quotient coefficient q_i the polynomial $q_i\mathbf{g}(X)$ must be subtracted from the dividend. The feedback connections in Figure 6.16 perform this subtraction. The difference between the leftmost p terms remaining in the dividend and the feedback terms $q_i\mathbf{g}(X)$ is formed on each shift of the circuit and appears as the contents of the register. At each shift of the register, the difference is shifted one stage; the highest-order term (which by construction is zero) is shifted out, while the next significant coefficient of $\mathbf{V}(X)$ is shifted in. After $m + 1$ total shifts into the register, the quotient has been serially presented at the output and the remainder resides in the register.

Example 6.9 Dividing Circuit

Use a dividing circuit of the form shown in Figure 6.16 to divide $\mathbf{V}(X) = X^3 + X^5 + X^6$ ($\mathbf{V} = 0\ 0\ 0\ 1\ 0\ 1\ 1$) by $\mathbf{g}(X) = (1 + X + X^3)$. Find the quotient and remainder terms. Compare the circuit implementation to the polynomial division steps performed by hand.

Solution

The dividing circuit needs to perform the following operation:

$$\frac{X^3 + X^5 + X^6}{1 + X + X^3} = \mathbf{q}(X) + \frac{\mathbf{p}(X)}{1 + X + X^3}$$

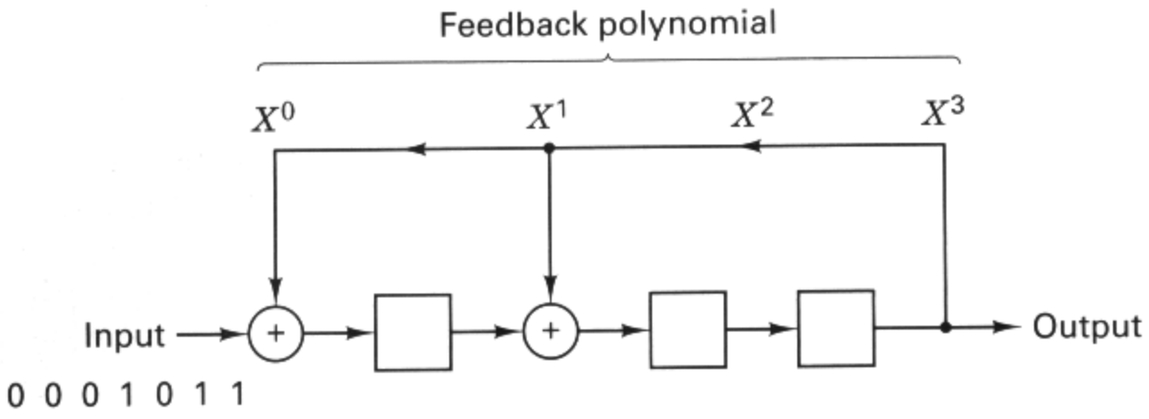


Figure 6.17 Dividing circuit for Example 6.9.

The required feedback shift register, following the general form of Figure 6.16, is shown in Figure 6.17. Assume that the register contents are initially zero. The operational steps of the circuit are as follows:

Input queue	Shift number	Register contents	Outout and Feedback
0 0 0 1 0 1 1	0	0 0 0	–
0 0 0 1 0 1	1	1 0 0	0
0 0 0 1 0	2	1 1 0	0
0 0 0 1	3	0 1 1	0
0 0 0	4	0 1 1	1
0 0	5	1 1 1	1
0	6	1 0 1	1
–	7	1 0 0	1

After the fourth shift, the quotient coefficients $\{q_i\}$ serially presented at the output are seen to be 1 1 1 1, or the quotient polynomial is $\mathbf{q}(X) = 1 + X + X^2 + X^3$. The remainder coefficients $\{p_i\}$ are 1 0 0, or the remainder polynomial $\mathbf{p}(X) = 1$. In summary, the circuit computation $\mathbf{V}(X)/\mathbf{g}(X)$ is seen to be

$$\frac{X^3 + X^5 + X^6}{1 + X + X^3} = 1 + X + X^2 + X^3 + \frac{1}{1 + X + X^3}$$

The polynomial division steps are as follows:

Output after shift number:

$$\begin{array}{rl} & \begin{array}{cccc} 4 & 5 & 6 & 7 \\ \downarrow & \downarrow & \downarrow & \downarrow \end{array} \\ & X^3 + X^2 + X + 1 \\ X^3 + X + 1 \;\big)\; & X^6 + X^5 \quad\quad + X^3 \\ & \underline{X^6 \quad\quad + X^4 + X^3} \quad \longleftarrow \text{feedback after 4th shift} \\ & \quad\quad X^5 + X^4 \quad \longleftarrow \text{register after 4th shift} \\ & \quad\quad \underline{X^5 \quad\quad + X^3 + X^2} \quad \longleftarrow \text{feedback after 5th shift} \\ & \quad\quad\quad\quad X^4 + X^3 + X^2 \quad \longleftarrow \text{register after 5th shift} \\ & \quad\quad\quad\quad \underline{X^4 \quad\quad + X^2 + X} \quad \longleftarrow \text{feedback after 6th shift} \\ & \quad\quad\quad\quad\quad\quad X^3 \quad\quad + X \quad \longleftarrow \text{register after 6th shift} \\ & \quad\quad\quad\quad\quad\quad \underline{X^3 \quad\quad + X + 1} \quad \longleftarrow \text{feedback after 7th shift} \\ & \quad\quad\quad\quad\quad\quad\quad\quad 1 \quad \longleftarrow \text{register after 7th shift} \\ & \quad\quad\quad\quad\quad\quad\quad\quad \text{(remainder)} \end{array}$$

6.7.5 Systematic Encoding with an ($n - k$)-Stage Shift Register

The encoding of a cyclic code in systematic form has been shown, in Section 6.7.3, to involve the computation of parity bits as the result of the formation of $X^{n-k}\mathbf{m}(X)$ modulo $\mathbf{g}(X)$, in other words, the *division* of an *upshifted* (right shifted) message polynomial by a generator polynomial $\mathbf{g}(X)$. The need for upshifting is to make room for the parity bits, which are appended to the message bits, yielding the code vector in systematic form. Upshifting the message bits by $n - k$ positions is a trivial operation and is not really performed as part of the dividing circuit. Instead, only the parity bits are computed; they are then placed in the appropriate location alongside the message bits. The parity polynomial is the *remainder* after dividing by the generator polynomial; it is available in the register after n shifts through the $(n - k)$-stage feedback register shown in Figure 6.17. Notice that the first $n - k$ shifts through the register are simply filling the register. We cannot have any feedback until the rightmost stage has been filled; we therefore can shorten the shifting cycle by loading the input data to the output of the last stage, as shown in Figure 6.18. Further, the feedback term into the leftmost stage is the sum of the input and the rightmost stage. We guarantee that this sum is generated by ensuring that $g_0 = g_{n-k} = 1$ for any generator polynomial $\mathbf{g}(X)$. The circuit feedback connections correspond to the coefficients of the generator polynomial, which is written as

$$\mathbf{g}(X) = 1 + g_1X + g_2X^2 + \cdots + g_{n-k-1}X^{n-k-1} + X^{n-k} \tag{6.66}$$

The following steps describe the encoding procedure used with the Figure 6.18 encoder:

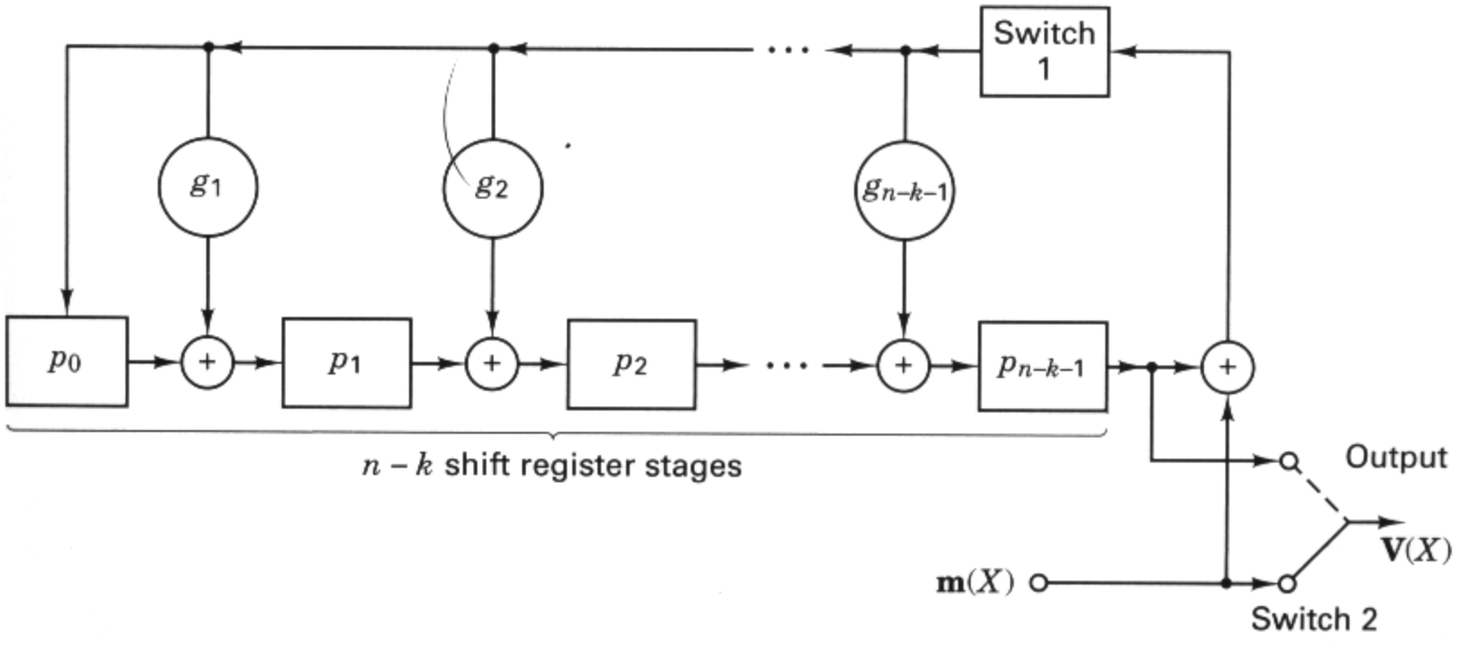


Figure 6.18 Encoding with an ($n - k$)-stage shift register.

1. Switch 1 is closed during the first k shifts, to allow transmission of the message bits into the $n - k$ stage encoding shift register.
2. Switch 2 is in the down position to allow transmission of the message bits directly to an output register during the first k shifts.
3. After transmission of the kth message bit, switch 1 is opened and switch 2 is moved to the up position.
4. The remaining $n - k$ shifts clear the encoding register by moving the parity bits to the output register.
5. The total number of shifts is equal to n, and the contents of the output register is the codeword polynomial $\mathbf{p}(X) + X^{n-k}\mathbf{m}(X)$.

Example 6.10 Systematic Encoding of a Cyclic Code

Use a feedback shift register of the form shown in Figure 6.18 to encode the message vector $\mathbf{m} = 1\ 0\ 1\ 1$ into a (7, 4) codeword using the generator polynomial $\mathbf{g}(X) = 1 + X + X^3$.

Solution

$$\mathbf{m} = 1\ 0\ 1\ 1$$

$$\mathbf{m}(X) = 1 + X^2 + X^3$$

$$X^{n-k}\mathbf{m}(X) = X^3\mathbf{m}(X) = X^3 + X^5 + X^6$$

$$X^{n-k}\mathbf{m}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{p}(X)$$

$$\mathbf{p}(X) = (X^3 + X^5 + X^6) \text{ modulo } (1 + X + X^3)$$

For the $(n - k) = 3$-stage encoding shift register shown in Figure 6.19, the operational steps are as follows:

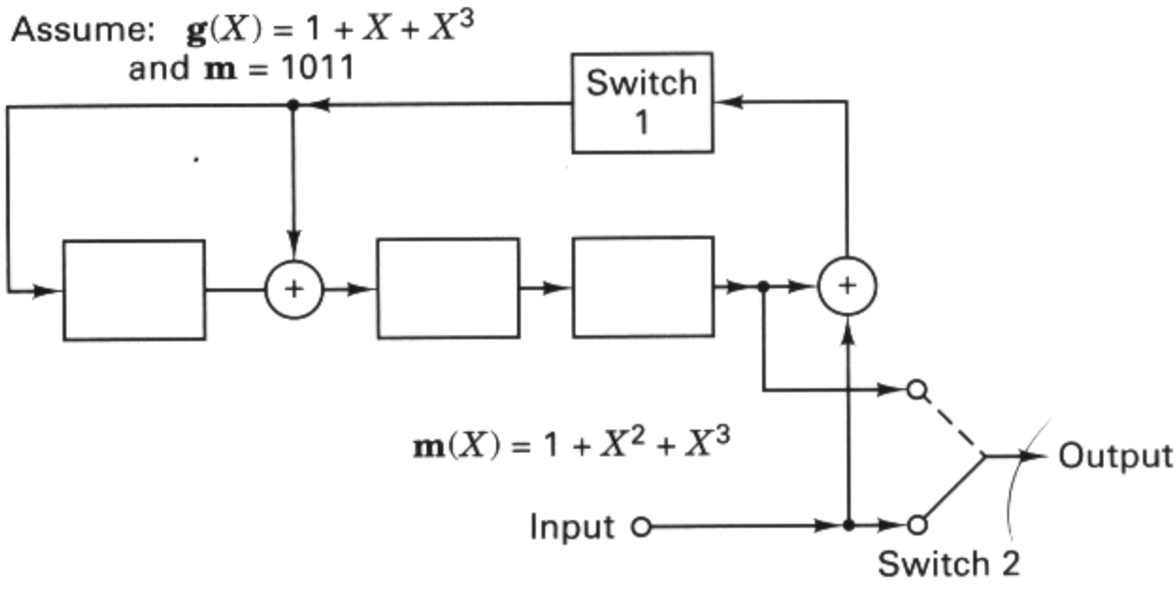


Figure 6.19 Example of encoding a (7, 4) cyclic code with an $(n - k)$-stage shift register.

Input queue	Shift number	Register contents	Output
1 0 1 1	0	0 0 0	–
1 0 1	1	1 1 0	1
1 0	2	1 0 1	1
1	3	1 0 0	0
–	4	1 0 0	1

After the fourth shift, switch 1 is opened, switch 2 is moved to the up position, and the parity bits contained in the register are shifted to the output. The output codeword is $\mathbf{U} = 1\,0\,0\,1\,0\,1\,1$, or in polynomial form, $\mathbf{U}(X) = 1 + X^3 + X^5 + X^6$.

6.7.6 Error Detection with an $(n - k)$-Stage Shift Register

A transmitted codeword may be perturbed by noise, and hence the vector received may be a corrupted version of the transmitted codeword. Let us assume that a codeword with polynomial representation $\mathbf{U}(X)$ is transmitted and that a vector with polynomial representation $\mathbf{Z}(X)$ is received. Since $\mathbf{U}(X)$ is a code polynomial, it must be a multiple of the generator polynomial $\mathbf{g}(X)$; that is,

$$\mathbf{U}(X) = \mathbf{m}(X)\mathbf{g}(X) \tag{6.67}$$

and $\mathbf{Z}(X)$, the corrupted version of $\mathbf{U}(X)$, can be written as

$$\mathbf{Z}(X) = \mathbf{U}(X) + \mathbf{e}(X) \tag{6.68}$$

where $\mathbf{e}(X)$ is the error pattern polynomial. The decoder tests whether $\mathbf{Z}(X)$ is a codeword polynomial, that is, whether it is divisible by $\mathbf{g}(X)$, with a zero remainder. This is accomplished by *calculating the syndrome* of the received polynomial. The syndrome $\mathbf{S}(X)$ is equal to the remainder resulting from dividing $\mathbf{Z}(X)$ by $\mathbf{g}(X)$; that is,

$$\mathbf{Z}(X) = \mathbf{q}(X)\mathbf{g}(X) + \mathbf{S}(X) \tag{6.69}$$

where $\mathbf{S}(X)$ is a polynomial of degree $n - k - 1$ or less. Thus, the syndrome is an $(n - k)$-tuple. By combing Equations (6.67) to (6.69), we obtain

$$\mathbf{e}(X) = [\mathbf{m}(X) + \mathbf{q}(X)]\mathbf{g}(X) + \mathbf{S}(X) \tag{6.70}$$

By comparing Equations (6.69) and (6.70), we see that the syndrome $\mathbf{S}(X)$, obtained as the remainder of $\mathbf{Z}(X)$ modulo $\mathbf{g}(X)$, is exactly the same polynomial obtained as the remainder of $\mathbf{e}(X)$ modulo $\mathbf{g}(X)$. Thus the syndrome of the received polynomial $\mathbf{Z}(X)$ contains the information needed for correction of the error pattern. The syndrome calculation is accomplished by a division circuit, almost identical to the encoding circuit used at the transmitter. An example of syndrome calculation with an $n - k$ shift register is shown in Figure 6.20 using the code vector generated in Example 6.10. Switch 1 is initially closed, and switch 2 is open. The received vector is shifted into the register input, with all stages initially set to zero. After the entire received vector has been entered into the shift register, the con-

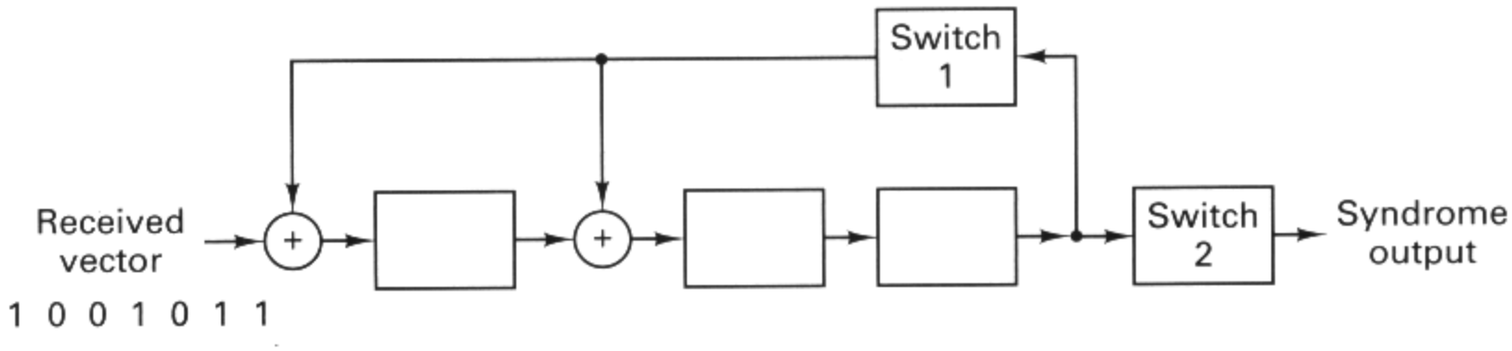


Figure 6.20 Example of syndrome calculation with an $(n - k)$-stage shift register.

tents of the register is the syndrome. Switch 1 is then opened and switch 2 is closed, so that the syndrome vector can be shifted out of the register. The operational steps of the decoder are as follows:

Input queue	Shift number	Register contents	
1 0 0 1 0 1 1	0	0 0 0	
1 0 0 1 0 1	1	1 0 0	
1 0 0 1 0	2	1 1 0	
1 0 0 1	3	0 1 1	
1 0 0	4	0 1 1	
1 0	5	1 1 1	
1	6	1 0 1	
–	7	0 0 0	Syndrome

If the syndrome is an all-zeros vector, the received vector is assumed to be a valid codeword. If the syndrome is a nonzero vector, the received vector is a perturbed codeword and errors have been detected; such errors can be corrected by adding the error vector (indicated by the syndrome) to the received vector, similar to the procedure described in Section 6.4.8. This method of decoding is useful for simple codes. More complex codes require the use of algebraic techniques to obtain practical decoders [6, 8].

6.8 WELL-KNOWN BLOCK CODES

6.8.1 Hamming Codes

Hamming codes are a simple class of block codes characterized by the structure

$$(n, k) = (2^m - 1, 2^m - 1 - m) \tag{6.71}$$

where $m = 2, 3, \ldots$. These codes have a minimum distance of 3 and thus, from Equations (6.44) and (6.47), they are capable of correcting all single errors or detecting all combinations of two or fewer errors within a block. Syndrome decoding is especially suited for Hamming codes. In fact, the syndrome can be formed to act

as a binary pointer to identify the error location [5]. Although Hamming codes are not very powerful, they belong to a very limited class of block codes known as *perfect* codes, described in Section 6.5.4.

Assuming hard decision decoding, the bit error probability can be written, from Equation (6.46), as

$$P_B \approx \frac{1}{n} \sum_{j=2}^{n} j \binom{n}{j} p^j (1-p)^{n-j} \tag{6.72}$$

where p is the channel symbol error probability (transition probability on the binary symmetric channel). In place of Equation (6.72) we can use the following equivalent equation. Its identity with Equation (6.72) is proven in Appendix D, Equation (D.16):

$$P_B \approx p - p(1-p)^{n-1} \tag{6.73}$$

Figure 6.21 is a plot of the decoded P_B versus channel-symbol error probability, illustrating the comparative performance for different types of block codes. For the

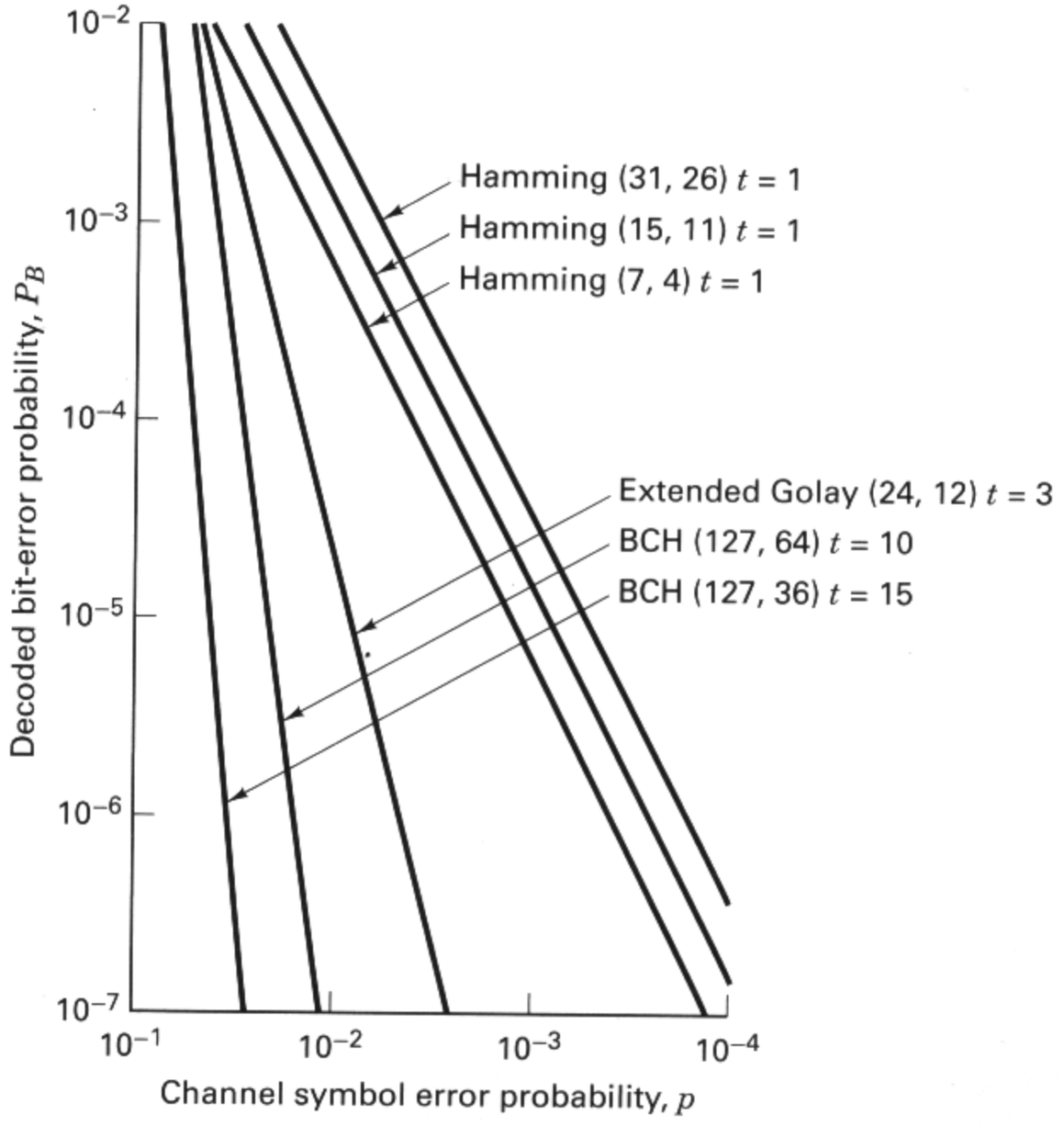


Figure 6.21 Bit error probability versus channel symbol error probability for several block codes.

Hamming codes, the plots are shown for $m = 3$, 4, and 5, or $(n, k) = (7, 4)$, (15, 11), and (31, 26). For performance over a Gaussian channel using coherently demodulated BPSK, we can express the channel symbol error probability in terms of E_c/N_0, similar to Equation (4.79), as

$$p = Q\left(\sqrt{\frac{2E_c}{N_0}}\right) \tag{6.74}$$

where E_c/N_0 is the code symbol energy per noise spectral density, and where $Q(x)$ is as defined in Equation (3.43). To relate E_c/N_0 to information bit energy per noise spectral density (E_b/N_0), we use

$$\frac{E_c}{N_0} = \left(\frac{k}{n}\right)\frac{E_b}{N_0} \tag{6.75}$$

For Hamming codes, Equation (6.75) becomes

$$\frac{E_c}{N_0} = \frac{2^m - 1 - m}{2^m - 1}\,\frac{E_b}{N_0} \tag{6.76}$$

Combining Equation (6.73), (6.74), and (6.76), P_B can be expressed as a function of E_b/N_0 for coherently demodulated BPSK over a Gaussian channel. The results are plotted in Figure 6.22 for different types of block codes. For the Hamming codes, plots are shown for $(n, k) = (7, 4)$, (15, 11), and (31, 26).

Example 6.11 Error Probability for Modulated and Coded Signals

A coded orthogonal BFSK modulated signal is transmitted over a Gaussian channel. The signal is noncoherently detected and hard-decision decoded. Find the decoded bit error probability if the coding is a Hamming (7, 4) block code and the received E_b/N_0 is equal to 20.

Solution

First we need to find E_c/N_0 using Equation (6.75):

$$\frac{E_c}{N_0} = \frac{4}{7}(20) = 11.43$$

Then, for coded noncoherent BFSK, we can relate the probability of a channel symbol error to E_c/N_0, similar to Equation (4.96), as follows

$$\begin{aligned} p &= \frac{1}{2}\exp\left(-\frac{E_c}{2N_0}\right) \\ &= \frac{1}{2}\exp\left(-\frac{11.43}{2}\right) = 1.6 \times 10^{-3} \end{aligned}$$

Using this result in Equation (6.73), we solve for the probability of a decoded bit error, as follows:

$$P_B \approx p - p(1-p)^6 \approx 1.6 \times 10^{-5}$$

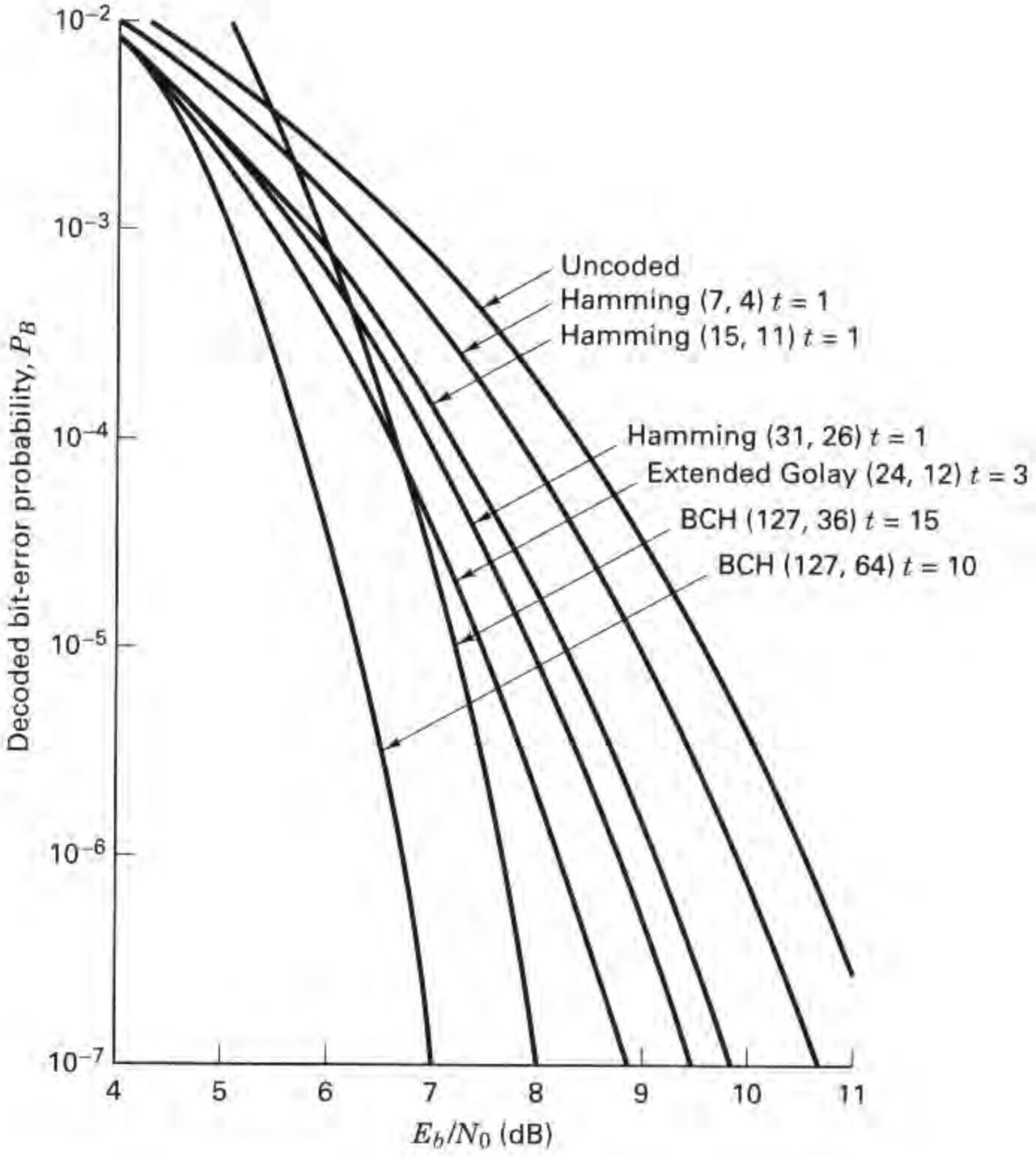


Figure 6.22 P_B versus E_b/N_0 for coherently demodulated BPSK over a Gaussian channel for several block codes.

6.8.2 Extended Golay Code

One of the more useful block codes is the binary (24, 12) *extended Golay code*, which is formed by adding an overall parity bit to the perfect (23, 12) code, known as the *Golay code*. This added parity bit increases the minimum distance $d_{\min}$ from 7 to 8 and produces a rate $\frac{1}{2}$ code, which is easier to implement (with regard to system clocks) than the rate 12/23 original Golay code. Extended Golay codes are considerably more powerful than the Hamming codes described in the preceding section. The price paid for the improved performance is a more complex decoder, a lower code rate, and hence a larger bandwidth expansion.

Since $d_{\min} = 8$ for the extended Golay code, we see from Equation (6.44) that the code is guaranteed to correct all triple errors. The decoder can additionally be designed to correct *some but not all* four-error patterns. Since only 16.7% of the four-error patterns can be corrected, the decoder, for the sake of simplicity, is usually designed to only correct three-error patterns [5]. Assuming hard decision decoding, the bit error probability for the extended Golay code can be written as a function of the channel symbol error probability p from Equation (6.46), as follows:

$$P_B \approx \frac{1}{24} \sum_{j=4}^{24} j \binom{24}{j} p^j (1-p)^{24-j} \tag{6.77}$$

The plot of Equation (6.77) is shown in Figure 6.21; the error performance of the extended Golay code is seen to be significantly better than that of the Hamming codes. Combining Equations (6.77), (6.74), and (6.75), we can relate P_B versus E_b/N_0 for coherently demodulated BPSK with extended Golay coding over a Gaussian channel. The result is plotted in Figure 6.22.

6.8.3 BCH Codes

Bose–Chadhuri–Hocquenghem (BCH) codes are a generalization of Hamming codes that allow multiple error correction. They are a *powerful class of cyclic codes* that provide a large selection of block lengths, code rates, alphabet sizes, and error-correcting capability. Table 6.4 lists some code generators $\mathbf{g}(x)$ commonly used for the construction of BCH codes [8] for various values of n, k, and t, up to a block length of 255. The coefficients of $\mathbf{g}(x)$ are presented as octal numbers arranged so that when they are converted to binary digits the rightmost digit corresponds to the zero-degree coefficient of $\mathbf{g}(x)$. From Table 6.4, one can easily verify a cyclic code property—the generator polynomial is of degree $n - k$. BCH codes are important because at block lengths of a few hundred, the BCH codes outperform all other block codes with the same block length and code rate. The most commonly used BCH codes employ a binary alphabet and a codeword block length of $n = 2^m - 1$, where $m = 3, 4, \ldots$.

The title of Table 6.4 indicates that the generators shown are for those BCH codes known as *primitive codes*. The term "primitive" is a number-theoretic concept requiring an algebraic development [7, 10–11], which is presented in Section 8.1.4. In Figures 6.21 and 6.22 are plotted error performance curves of two BCH codes (127, 64) and (127, 36), to illustrate comparative performance. Assuming hard decision decoding, the P_B versus channel error probability is shown in Figure 6.21. The P_B versus E_b/N_0 for coherently demodulated BPSK over a Gaussian channel is shown in Figure 6.22. The curves in Figure 6.22 seem to depart from our expectations. They each have the same block size, yet the more redundant (127, 36) code does not exhibit as much coding gain as does the less redundant (127, 64) code. It has been shown that a relatively broad maximum of coding gain versus code rate for fixed n occurs roughly between coding rates of $\frac{1}{3}$ and $\frac{3}{4}$ for BCH codes [12]. Performance over a Gaussian channel degrades substantially at very high or very low rates [11].

Figure 6.23 represents computed performance of BCH codes [13] using coherently demodulated BPSK with both *hard-* and *soft-decision decoding*. Soft-decision decoding is not usually used with block codes because of its complexity. However, whenever it is implemented, it offers an approximate 2-dB coding gain over hard-decision decoding. For a given code rate, the decoded error probability is known to improve with increasing block length n [4]. Thus, for a given code rate, it is

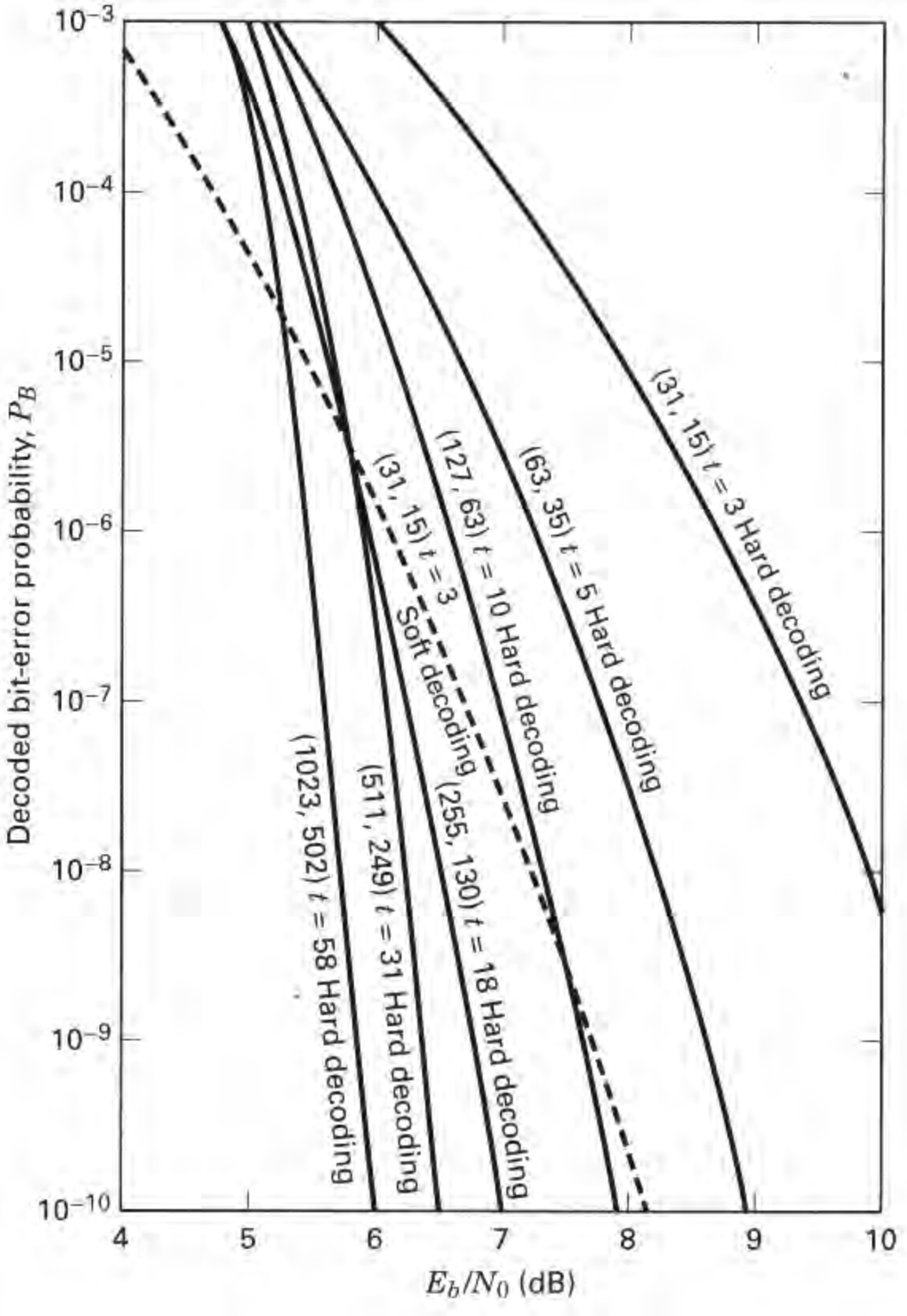


Figure 6.23 P_B versus E_b/N_0 for coherently demodulated BPSK over a Gaussian channel using BCH codes. (Reprinted with permission from L. J. Weng, "Soft and Hard Decoding Performance Comparisons for BCH Codes," *Proc. Int. Conf. Commun.*, 1979, Fig. 3, p. 25.5.5. © 1979 IEEE.)

TABLE 6.4 Generators of Primitive BCH Codes

n	k	t	$\mathbf{g}(x)$
7	4	1	13
15	11	1	23
	7	2	721
	5	3	2467
31	26	1	45
	21	2	3551
	16	3	107657
	11	5	5423325
	6	7	313365047
63	57	1	103
	51	2	12471
	45	3	1701317
	39	4	166623567
	36	5	1033500423
	30	6	157464165547
	24	7	17323260404441
	18	10	1363026512351725
	16	11	6331141367235453
	10	13	472622305527250155
	7	15	5231045543503271737
255	171	11	154162142123423560770616306 37
	163	12	75004155100756025515747245146 01
	155	13	37575130054076650157225064646776 33
	147	14	16421301735371655253041653054410117 11
	139	15	46140173206017556157072273024745356744 5
	131	18	21571333147151015126125027744214202416 5471
	123	19	12061450522420660037172103265161412262 72506267
	115	21	60526665572100247263636404600276352556 313472737
	107	22	22205772322066256312417300235347420176 574750154441
	99	23	10656667253473174222741416201574332252 411076432303431
	91	25	67502650303274441727236317247325110755 07622720724344561
	87	26	11013676341474323643523163430717204626 72545273311721317
	79	27	66700035637657500020270344207366174621 532671176654134235 5

127	120	1	211			
	113	2	41567	71	29	24024710520644321515554172112331163205444250362557643221706035
	106	3	11554743			
	99	4	3447023271	63	30	10754475055163544325315217357707003666111726455267613656702543301
	92	5	624730022327			
	85	6	130704476322273	55	31	73154252035011001330152753060320543254143267550105570444260354736 17
	78	7	26230002166130115			
	71	9	6255010713253127753	47	42	25335420170626465630330413774062331751233341454460450050660245525431 73
	64	10	1206534025570773100045			
	57	11	335265252505705053517721	45	43	15202056055234161131101346376423701563670024470762373033202157025051541
	50	13	544465125233140124215014 21			
	43	14	177217722136512275212205743 43	37	45	51363302550670074141774472454375304207357061743234323476443547374030440035
	36	15	31460746665220750447645747217 35			
	29	21	403114461367670603667530141176155	29	47	30257155366730714655270640123613771153422423242011741140602547574104035650 37
	22	23	123376070404722522435445626637647043			
	15	27	220570424456045547705230137622176043 53			
	8	31	704726405275103065147622427156773313021 7	21	55	12562152570603326560017731536076121032273414056530745425211531216144665134737 25
255	247	1	435			
	239	2	267543			
	231	3	156720665	13	59	46417320050525645444265737142500660043306774454765614031746772135702613446050054 7
	223	4	75626641375			
	215	5	23157564726421			
	207	6	16176560567636227	9	63	15726025217472463201031043255355134614162367212044074545112766115547705561677516057
	199	7	7633031270420722341			
	191	8	2663470176115333714567			
	187	9	52755313540001322236351			
	179	10	22624710717340432416300455			

Source: Reprinted with permission from "Table of Generators for BCH Codes," *IEEE Trans. Inf. Theory,* vol. IT10, no. 4, Oct. 1964, p. 391. © 1964 IEEE.

interesting to consider the block length that would be required for the hard-decision-decoding performance to be comparable to the soft-decision-decoding performance. In Figure 6.23, the BCH codes shown all have code rates of approximately $\frac{1}{2}$. From the figure [13] it appears that for a fixed code rate, the hard-decision-decoded BCH code of length 8 times n or longer has a better performance (for P_B of about 10^{-6} or less) than that of a soft-decision-decoded BCH code of length n. One special subclass of the BCH codes (the discovery of which preceded the BCH codes) is the particularly useful *nonbinary* set called *Reed-Solomon* codes. They are described in Section 8.1.

6.9 CONCLUSION

In this chapter we have explored the general goals of channel coding, leading to improved performance (error probability, E_b/N_0, or capacity) at a cost in bandwidth. We partitioned channel coding into two study groups: waveform coding and structured sequences. Waveform coding represents a transformation of waveforms into improved waveforms, such that the distance properties are improved over those of the original waveforms. Structured sequences involve the addition of redundant digits to the data, such that the redundant digits can then be employed for detecting and/or correcting specific error patterns.

We also closely examined linear block codes. Geometric analogies can be drawn between the coding and modulation disciplines. They both seek to pack the signal space efficiently and to maximize the distance between signals in the signaling set. Within block codes, we looked at cyclic codes, which are relatively easy to implement using modern integrated circuit techniques. We considered the polynomial representation of codes and the correspondence between the polynomial structure, the necessary algebraic operations, and the hardware implementation. Finally, we looked at performance details of some of the well-known block codes. Other coding subjects are treated in later chapters. In Chapter 7 we study the large class of convolutional codes; in Chapter 8 we discuss Reed–Solomon codes, concatenated codes, and turbo codes; and in Chapter 9 we examine trellis-coded modulation.

REFERENCES

1. Viterbi, A. J., "On Coded Phase-Coherent Communications," *IRE Trans. Space Electron. Telem.*, vol. SET7, Mar. 1961, pp. 3–14.
2. Lindsey, W. C., and Simon, M. K., *Telecommunication Systems Engineering*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1973.
3. Proakis, J. G., *Digital Communications*, McGraw-Hill Book Company, New York, 1983.
4. Lin, S., and Costello, D. J., Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983.
5. Odenwalder, J. P., *Error Control Coding Handbook*, Linkabit Corporation, San Diego, Calif., July 15, 1976.

6. Blahut, R. E., *Theory and Practice of Error Control Codes,* Addison-Wesley Publishing Company, Inc., Reading, Mass, 1983.
7. Peterson, W. W., and Weldon, E. J., *Error Correcting Codes,* 2nd ed., The MIT Press, Cambridge, Mass., 1972.
8. Blahut, R. E., "Algebraic Fields, Signal Processing, and Error Control," *Proc. IEEE,* vol. 73, May 1985, pp. 874–893.
9. Stenbit, J. P., "Table of Generators for Bose–Chadhuri Codes, *IEEE Trans. Inf. Theory,* vol. IT10, no. 4, Oct. 1964, pp. 390–391.
10. Berlekamp, E. R., *Algebraic Coding Theory,* McGraw-Hill Book Company, New York, 1968.
11. Clark, G. C., Jr., and Cain, J. B. *Error-Correction Coding for Digital Communications,* Plenum Press, New York, 1981.
12. Wozencraft, J. M., and Jacobs, I. M., *Principles of Communication Engineering,* John Wiley & Sons, Inc., New York, 1965.
13. Weng, L. J., "Soft and Hard Decoding Performance Comparisons for BCH Codes," *Proc. Int. Conf. Commun.,* 1979, pp. 25.5.1–25.5.5

PROBLEMS

6.1. Design an (n, k) single-parity code that will detect all 1-, 3-, 5-, and 7-error patterns in a block. Show the values of n and k, and find the probability of an undetected block error if the probability of channel symbol error is 10^{-2}.

6.2. Calculate the probability of message error for a 12-bit data sequence encoded with a (24, 12) linear block code. Assume that the code corrects all 1-bit and 2-bit error patterns and assume that it corrects no error patterns with more than two errors. Also, assume that the probability of a channel symbol error is 10^{-3}.

6.3. Consider a (127, 92) linear block code capable of triple error corrections.
- **(a)** What is the probability of message error for an uncoded block of 92 bits if the channel symbol error probability is 10^{-3}?
- **(b)** What is the probability of message error when using the (127, 92) block code if the channel symbol error probability of 10^{-3}?

6.4. Calculate the improvement in probability of message error relative to an uncoded transmission for a (24, 12) double-error-correcting linear block code. Assume that coherent BPSK modulation is used and that the received $E_b/N_0 = 10$ dB.

6.5. Consider a (24, 12) linear block code capable of double-error corrections. Assume that a noncoherently detected binary orthogonal frequency-shift keying (BFSK) modulation format is used and that the received $E_b/N_0 = 14$ dB.
- **(a)** Does the code provide any improvement in probability of message error? If it does, how much? If it does not, explain why not.
- **(b)** Repeat part (a) with $E_b/N_0 = 10$ dB.

6.6. The telephone company uses a "best-of-five" encoder for some of its digital data channels. In this system every data bit is repeated five times, and at the receiver, a majority vote decides the value of each data bit. If the uncoded probability of bit error is 10^{-3}, calculate the decoded bit-error probability when using such a best-of-five code.

6.7. The minimum distance for a particular linear block code is 11. Find the maximum error-correcting capability, the maximum error-detecting capability, and the maximum erasure-correcting capability in a block length.

6.8. Consider a (7, 4) code whose generator matrix is

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(a) Find all the codewords of the code.
(b) Find **H**, the parity-check matrix of the code.
(c) Compute the syndrome for the received vector 1 1 0 1 1 0 1. Is this a valid code vector?
(d) What is the error-correcting capability of the code?
(e) What is the error-detecting capability of the code?

6.9. Consider a systematic block code whose parity-check equations are

$$p_1 = m_1 + m_2 + m_4$$
$$p_2 = m_1 + m_3 + m_4$$
$$p_3 = m_1 + m_2 + m_3$$
$$p_4 = m_2 + m_3 + m_4$$

where m_i are message digits and p_i are check digits.
(a) Find the generator matrix and the parity-check matrix for this code.
(b) How many errors can the code correct?
(c) Is the vector 10101010 a codeword?
(d) Is the vector 01011100 a codeword?

6.10. Consider the linear block code with the codeword defined by

$$\mathbf{U} = m_1 + m_2 + m_4 + m_5,\ m_1 + m_3 + m_4 + m_5,\ m_1 + m_2 + m_3 + m_5,$$
$$m_1 + m_2 + m_3 + m_4,\ m_1,\ m_2,\ m_3,\ m_4,\ m_5$$

(a) Show the generator matrix.
(b) Show the parity-check matrix.
(c) Find n, k, and $d_{\min}$.

6.11. Design an $(n, k) = (5, 2)$ linear block code.
(a) Choose the codewords to be in systematic form, and choose them with the goal of maximizing $d_{\min}$.
(b) Find the generator matrix for the codeword set.
(c) Calculate the parity-check matrix.
(d) Enter all of the n-tuples into a standard array.
(e) What are the error-correcting and error-detecting capabilities of the code?
(f) Make a syndrome table for the correctable error patterns.

6.12. Consider the (5, 1) repetition code, which consists of the two codewords 00000 and 11111, corresponding to message 0 and 1, respectively. Derive the standard array for this code. Is this a perfect code?

6.13. Design a (3, 1) code that will correct all single-error patterns. Choose the codeword set and show the standard array.

6.14. Is a (7, 3) code a perfect code? Is a (7, 4) code a perfect code? Is a (15, 11) code a perfect code? Justify your answers.

6.15. A (15, 11) linear block code can be defined by the following parity array:

$$\mathbf{P} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

(a) Show the parity-check matrix for this code.

(b) List the coset leaders from the standard array. Is this code a perfect code? Justify your answer.

(c) A received vector is $\mathbf{V} = 0\ 1\ 1\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 1\ 1$. Compute the syndrome. Assuming that a single bit error has been made, find the correct codeword.

(d) How many erasures can this code correct? Explain.

6.16. Is it possible that a nonzero error pattern can produce a syndrome of $\mathbf{S} = \mathbf{0}$? If yes, how many such error patterns can give this result for an (n, k) code? Use Figure 6.11 to justify your answer.

6.17. Determine which, if any, of the following polynomials can generate a cyclic code with codeword length $n \leq 7$. Find the (n, k) values of any such codes that can be generated.

(a) $1 + X^3 + X^4$

(b) $1 + X^2 + X^4$

(c) $1 + X + X^3 + X^4$

(d) $1 + X + X^2 + X^4$

(e) $1 + X^3 + X^5$

6.18. Encode the message 1 0 1 in systematic form using polynomial division and the generator $\mathbf{g}(X) = 1 + X + X^2 + X^4$.

6.19. Design a feedback shift register encoder for an (8, 5) cyclic code with a generator $\mathbf{g}(x) = 1 + X + X^2 + X^3$. Use the encoder to find the codeword for the message 1 0 1 0 1 in systematic form.

6.20. In Figure P6.1 the signal is differentially coherent PSK (DPSK), the encoded symbol rate is 10,000 code symbols per second, and the decoder is a single-error-correcting (7, 4) decoder. Is a predetection signal-to-noise spectral density ratio of $P_r/N_0 = 48$ dBW sufficient to provide a probability of message error of 10^{-3} at the output?

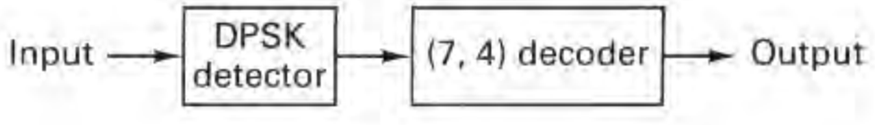


Figure P6.1

Justify your answer. Assume that a message block contains 4 data bits and that any single-error pattern in a block length of 7 bits can be corrected.

6.21. A (15, 5) cyclic code has a generator polynomial as follows:

$$\mathbf{g}(X) = 1 + X + X^2 + X^5 + X^8 + X^{10}$$

(a) Draw a diagram of an encoder for this code.

(b) Find the code polynomial (in systematic form) for the message $\mathbf{m}(X) = 1 + X^2 + X^4$.

(c) Is $\mathbf{V}(X) = 1 + X^4 + X^6 + X^8 + X^{14}$ a code polynomial in this system? Justify your answer.

6.22. Consider the (15, 11) cyclic code generated by $\mathbf{g}(X) = 1 + X + X^4$.

(a) Devise a feedback register encoder and decoder for this code.

(b) Illustrate the encoding procedure with the message vector 11001101011 by listing the states of the register (the rightmost bit is the earliest bit).

(c) Repeat part (b) for the decoding procedure.

6.23. For a fixed probability of channel symbol error, the probability of bit error for a Hamming (15, 11) code is worse than that for a Hamming (7, 4) code. Explain why. What, then, is the advantage of the (15, 11) code? What basic trade-off is involved?

6.24. A (63, 36) BCH code can correct five errors. Nine blocks of a (7, 4) code can correct nine errors. Both codes have the same code rate.

(a) The (7, 4) code can correct more errors. Is it more powerful? Explain.

(b) Compare the two codes when five errors occur randomly in 63 bits.

6.25. Information from a source is organized in 36-bit messages that are to be transmitted over an AWGN channel using noncoherently detected BFSK modulation.

(a) If no error control coding is used, compute the E_b/N_0 required to provide a message error probability of 10^{-3}.

(b) Consider the use of a (127, 36) linear block code (minimum distance is 31) in the transmission of these messages. Compute the coding gain for this code for a message error probability of 10^{-3}. (*Hint:* The coding gain is defined as the difference between the E_b/N_0 required without coding and the E_b/N_0 required with coding.)

6.26. (a) Consider a data sequence encoded with a (127, 64) BCH code and then modulated using coherent 16-ary PSK. If the received E_b/N_0 is 10 dB, find the MPSK probability of symbol error, the probability of code-bit error (assuming that a Gray code is used for symbol-to-bit assignment), and the probability of information-bit error.

(b) For the same probability of information-bit error found in part (a), determine the value of E_b/N_0 required if the modulation in part (a) is changed to coherent orthogonal 16-ary FSK. Explain the difference.

6.27. A message consists of English text (assume that each word in the message contains six letters). Each letter is encoded using the 7-bit ASCII character code. Thus, each word of text consists of a 42-bit sequence. The message is to be transmitted over a channel having a symbol error probability of 10^{-3}.

(a) What is the probability that a word will be received in error?

(b) If a repetition code is used such that each letter in each word is repeated three times, and at the receiver, majority voting is used to decode the message, what is the probability that a decoded word will be in error?

(c) If a (126, 42) BCH code with error-correcting capability of $t = 14$ is used to encode each 42-bit word, what is the probability that a decoded word will be in error?

(d) For a real system, it is not fair to compare uncoded versus coded message error performance on the basis of a fixed probability of channel symbol error, since this implies a fixed level of received E_c/N_0 for all choices of coding (or lack of coding). Therefore, repeat parts (a), (b), and (c) under the condition that the channel symbol error probability is determined by a received E_b/N_0 of 12 dB, where E_b/N_0 is the information bit energy per noise spectral density. Assume that the information rate must be the same for all choices of coding or lack of coding. Also assume that noncoherent orthogonal binary FSK modulation is used over an AWGN channel.

(e) Discuss the relative error performance capabilities of the above coding schemes under the two postulated conditions—fixed channel symbol error probability, and fixed E_b/N_0. Under what circumstances can a repetition code offer error performance improvement? When will it cause performance degradation?

6.28. A 5-bit data sequence is transformed into an orthogonal coded sequence using a Hadamard matrix. Coherent detection is performed over a codeword interval of time, as shown in Figure 6.5. Compute the coding gain relative to transmitting the data one bit at a time using BPSK.

6.29. For the (8, 2) code described in Section 6.6.3, verify that the values given for the generator matrix, the parity-check matrix, and the syndrome vectors for each of the cosets from 1 through 10, are valid.

6.30. Using exclusive-OR gates and AND gates, implement a decoder circuit, similar to the one shown in Figure 6.12, that will perform error correction for all single-error patterns of the (8, 2) code described by the coset leaders 2 through 9 in Figure 6.15.

6.31. Explain in detail how you could use exclusive-OR gates and AND gates to implement a decoder circuit, similar to the one shown in Figure 6.12, that performs error correction for all single and double-error patterns of the (8, 2) code, and performs error detection for the triple-error patterns (coset leaders or rows 38 through 64).

6.32. Verify that all of the BCH codes of length $n = 31$, shown in Table 6.4, meet the Hamming bound and the Plotkin bound.

6.33. When encoding an all-zeros message block, the result is an all-zeros codeword. It is generally undesirable to transmit long runs of such zeros. One cyclic encoding echnique that avoids such transmissions involves preloading the shift-register stages with ones instead of zeros, prior to encoding. The resulting "pseudoparity" is then guaranteed to contain some ones. At the decoder, there needs to be a reversal of the pseudoparity before the decoding operation starts. Devise a general scheme for reversing the pseudo-parity at any such cyclic decoder. Use a (7, 4) BCH encoder, preloaded with ones, to encode the message 1 0 1 1 (right-most bit is earliest). Then demonstrate that your reversal scheme, which is applied prior to decoding, yields the correct decoded message.

6.34. **(a)** Using the generator polynomial for the (15, 5) cyclic code in Problem 6.21, encode the message sequence 1 1 0 1 1 in systematic form. Show the resulting codeword polynomial. What property characterizes the degree of the generator polynomial?

(b) Consider that the received codeword is corrupted by an error pattern $\mathbf{e}(X) = X^8 + X^{10} + X^{13}$. Show the corrupted codeword polynomial.

(c) Form the syndrome polynomial by using the generator and received-codeword polynomials.

(d) Form the syndrome polynomial by using the generator and error-pattern polynomials, and verify that this is the same syndrome computed in part (c).

(e) Explain why the syndrome computations in parts (c) and (d) must yield identical results.

(f) Using the properties of the standard array of a (15, 5) linear block code, find the maximum amount of error correction possible for a code with these parameters. Is a (15, 5) code a perfect code?

(g) If we want to implement the (15, 5) cyclic code to simultaneously correct two erasures and still perform error correction, how much error correction would have to be sacrificed?

QUESTIONS

6.1. Describe four types of trade-offs that can be accomplished by using an error-correcting code. (See Section 6.3.4.)

6.2. In a *real-time* communication system, achieving coding gain by adding redundancy costs *bandwidth*. What is the usual cost for achieving coding gain in a *non-real-time* communication system? (See Section 6.3.4.2.)

6.3. In a real-time communication system, added redundancy means faster signaling, less energy per channel symbol, and more errors out of the demodulator. In the face of such degradation, explain how coding gain is achieved. (See Example 6.2.)

6.4. Why do error-correcting codes typically yield error-performance degradation at low values of E_b/N_0? (See Section 6.3.4.6.)

6.5. Describe the process of syndrome testing, error detection and correction in the context of a medical analogy. (See Section 6.4.8.4.)

6.6. Of what use is the *standard array* in understanding a block code, and in evaluating its capability? (See Section 6.6.5.)

EXERCISES

Using the Companion CD, run the exercises associated with Chapter 6.